Java SE
程序设计高级教程

青岛东合信息技术有限公司　青岛海尔软件有限公司　编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书在理解 Java 面向对象编程思想的基础上，深入讲解了 Java 的高级应用。全书共有 9 章，分别介绍了线程、网络编程、Swing 图形界面（包括两章）、事件处理、JDBC、RMI、国际化和正则表达式。书中涉及 Java GUI 设计及事件处理技巧，通过多线程实现 Java 多任务处理，通过 Socket 编程体验 Java 对网络编程的支持，通过 JDBC 实现 Java 访问数据库及 JDBC 的规范和技巧，系统地介绍 Java 远程调用 RMI 技术及安全策略，介绍正则表达式的原理及 Java 对正则表达式的支持等内容。

本书重点突出、偏重应用，结合理论篇的实例和实践篇对贯穿案例的讲解、剖析及实现，使读者能迅速理解并掌握知识，全面提高动手能力。

本书适应面广，可作为本科计算机科学与技术、软件外包专业、高职高专计算机软件、计算机网络、计算机信息管理、电子商务和经济管理等专业的程序设计课程的教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

Java SE 程序设计高级教程 / 青岛东合信息技术有限公司，青岛海尔软件有限公司编著.
— 北京：电子工业出版社，2010.8
ISBN 978-7-121-11268-3

Ⅰ. ①J… Ⅱ. ①青… ②青… Ⅲ. ①JAVA 语言—程序设计—高等学校—教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2010)第 128285 号

责任编辑：张月萍
印　　刷：北京天宇星印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：21　　字数：537.6 千字
印　　次：2010 年 8 月第 1 次印刷
定　　价：46.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件到 dbqq@phei.com.cn。

服务热线：（010）88258888。

编　委　会

主　编：邵峰晶

副主编：于忠清　　韩敬海

特约策划人：吕　蕾

编　委：曹宝香　　崔文善　　肖孟强

薛庆文　　庞善臣　　高仲合

陈龙猛　　张　磊　　吴海峰

刘宝旨　　钟岱辉

前　言

随着 IT 产业的迅猛发展，企业对应用型人才的需求越来越大。“全面贴近企业需求，无缝打造专业实用人才”是目前高校计算机专业教育的革新方向。

该系列教材是面向高等院校软件专业方向的标准化教材。本教材充分结合软件企业的用人需求，经过了充分的调研和论证，并参照多所高校一线专家的意见，具有系统性、实用性等特点。旨在使读者在系统掌握软件开发知识的同时，着重培养其综合应用能力和解决问题的能力。

该系列教材具有如下几个特色。

1. 以应用型人才为导向来培养学生

强调实践：本系列教材以应用型软件及外包人才为培养目标，在原有体制教育的基础上对课程进行了改革，强化“应用型”技术的学习。使学生在经过系统、完整的学习后能够达到如下要求：

- 具备软件开发工作所需的理论知识和操作技能，能熟练进行编码工作，并掌握软件开发过程的规范；
- 具备一定的项目经验，包括代码的调试、文档编写和软件测试等内容；
- 相当于一年的软件开发经验。

2. 以实用技能为核心来组织教学

二八原则：遵循企业生产过程中的“二八原则”，即企业生产过程中 80%的时间在使用 20%的核心技术，强调核心教学，即学生在学校用 80%的学习时间来掌握企业中所用到的核心技术，从而保证对企业常用技术的掌握。教材内容精而专，同时配以知识拓展和拓展练习，以满足不同层次的教学和学习需求。

3. 以新颖的教材架构来引导学习

自成体系：本系列教材采用的教材架构打破了传统的以知识为标准编写教材的方法，采用“全真案例”和“任务驱动”的组织模式。

- **理论篇**：即最小教学集，包含了“二八原则”中提到的常用技术，以任务驱动引导知识点的学习，所选任务不但典型、实用，而且具有很强的趣味性和可操作性，引导学生循序渐进地理解和掌握这些知识和技能，培养学生的逻辑思维能力，掌握利用开发语言进行程序设计的必备知识和技巧。
- **实践篇**：多点于一线，以完整的具体案例贯穿始终，力求使学生在动手实践的过程中，加深课程内容的理解，培养学生独立思考和解决问题的能力，并配备相关知识的拓展讲解和拓展练习，拓宽学生的知识面。
- **结构灵活**：本系列教材在内容设置上借鉴了软件开发中“低耦合高内聚”的设计理念，组织架构上遵循软件开发中的 MVC 理念，即在课程的实施过程中各高校可根据自身的实际情况（课程配比、时间安排、学生水平、教学情况等），在保证最小教学集的前提下对整个课程体系进行横向（章节内容）和纵向（章节）的裁剪。

4. 提供全面的教辅产品来辅助教学实施

为充分体现“实境耦合”的教学模式，方便教学实施，我们另外还开发了可配套使用的项目实训教材和全套教辅产品，可供各院校选购：

项目篇：多线于一面，以辅助教材的形式，提供适应当前课程（及先行课程）的综合项目，遵循软件开发过程，进行讲解、分析、设计、指导，注重工作过程的系统性，培养学生解决实际问题的能力，是实施“实境”教学的关键环节。

立体配套：为适应教学模式和教学方法的改革，本系列教材提供完备的教辅产品，主要包括教学指导、实验指导、电子课件、习题集、实践案例等内容，并配以相应的网络教学资源。教学实施方面，提供全方位的解决方案（课程体系解决方案、实训解决方案、教师培训解决方案和就业指导解决方案等），以适应软件开发教学过程的特殊性。

本系列教材由青岛海尔软件有限公司和青岛东合信息技术有限公司共同研制，历时两年，参与编著的有张磊、丁春强、赵克玲、高峰、张旭平、孙更新、宾晟、胡芳、侯天超、邵峰晶、于忠清、韩敬海、曹宝香、崔文善、肖孟强等。本书的特约策划人为吕蕾女士。参与本书编写工作的单位还有：青岛农业大学、潍坊学院、青岛大学、山东科技大学、曲阜师范大学、济宁学院、中国海洋大学、青岛科技大学、济宁医学院等高校，期间得到了各合作院校专家及一线教师的大力支持和协作。在此技术丛书出版之际要特别感谢给予我们开发团队大力支持和帮助的领导及同事，感谢合作院校的师生给予我们的支持和鼓励，更要感谢开发团队每一位成员所付出的艰辛劳动。如有意见及建议，请发邮件至 iTeacher@haiersoft.com.cn。

iTeacher@ 教研组

2010 年 7 月

目　　录

理论篇 1

第 1 章　线程 2
1.1　线程基础 4
1.1.1　线程概述 4
1.1.2　Java 线程模型 4
1.2　线程使用 5
1.2.1　创建线程 5
1.2.2　线程状态 8
1.2.3　线程优先级 12
1.2.4　线程组 14
1.3　多线程 15
1.3.1　多线程概述 15
1.3.2　线程同步 16
1.3.3　线程通信 19
1.3.4　死锁 22
小结 24
练习 25

第 2 章　网络编程 26
2.1　网络基础 28
2.1.1　网络类型 28
2.1.2　TCP/IP 协议 28
2.1.3　IP 地址 30
2.1.4　端口 31
2.1.5　域名与 DNS 31
2.2　网络 API 32
2.2.1　InetAddress 类 32
2.2.2　URL 类 34
2.2.3　URLConnection 类 36
2.3　基于 TCP 的网络编程 37
2.3.1　Socket 类 38
2.3.2　ServerSocket 类 39
2.3.3　C/S 实例 40
2.3.4　多线程 Socket 通信 43
小结 45

练习 45
第 3 章 Swing 图形界面（1） 47
3.1 Swing 概述 49
3.1.1 Swing 简介 49
3.1.2 Swing 的结构 49
3.2 容器 50
3.2.1 顶层容器 50
3.2.2 中间容器 52
3.3 布局 53
3.3.1 FlowLayout 53
3.3.2 BorderLayout 55
3.3.3 GridLayout 56
3.3.4 CardLayout 58
3.3.5 NULL 布局 60
3.4 Swing 常用组件 61
3.4.1 按钮 61
3.4.2 标签 62
3.4.3 图标 62
3.4.4 文本组件 63
3.4.5 复选框 64
3.4.6 单选按钮 65
3.4.7 列表框 66
3.4.8 组合框 67
3.5 Swing 组件示例 67
3.5.1 登录界面 67
3.5.2 注册界面 68
小结 71
练习 72
第 4 章 事件处理 73
4.1 事件概述 75
4.1.1 Java 事件处理机制 75
4.1.2 事件处理要点 75
4.1.3 Java 事件体系结构 76
4.2 事件处理 76
4.2.1 事件类 76
4.2.2 监听接口 77
4.3 事件示例 78
4.3.1 行为事件处理示例 78
4.3.2 选项事件处理示例 80
4.3.3 键盘事件处理 82
4.3.4 鼠标事件处理 84
4.4 适配器 85
4.5 多事件处理 87

小结 90
练习 90

第 5 章　Swing 图形界面（2） 92
5.1　菜单 94
5.1.1　菜单栏（JMenuBar） 94
5.1.2　菜单（JMenu） 94
5.1.3　菜单项（JMenuItem） 94
5.1.4　菜单示例 95
5.1.5　弹出式菜单 97
5.2　工具栏 99
5.3　对话框 100
5.3.1　标准对话框 100
5.3.2　对话框 104
5.3.3　文件对话框 107
5.3.4　颜色对话框 111
5.4　JTable 类 114
5.4.1　表格 114
5.4.2　表格模型 115
5.4.3　表格列模型 115
5.4.4　表格选择模式 116
5.5　JTree 类 119
5.5.1　树 119
5.5.2　树模型 120
5.5.3　树节点 120
5.5.4　树事件 121
小结 124
练习 124

第 6 章　JDBC 125
6.1　JDBC 基础 127
6.1.1　JDBC 概述 127
6.1.2　JDBC 结构 127
6.1.3　JDBC 类型 128
6.1.4　JDBC 与 ODBC 129
6.2　访问数据库 130
6.2.1　连接数据库 131
6.2.2　连接实例 134
6.3　操作数据库 136
6.3.1　Statement 接口 136
6.3.2　PreparedStatement 接口 140
6.3.3　CallableStatement 接口 142
6.4　集元数据 145
6.4.1　DatabaseMetaData 接口 145
6.4.2　ResultSetMetaData 接口 147

6.5 事务操作 148
6.5.1 事务 148
6.5.2 保存点 150
小结 151
练习 152

第 7 章 RMI 153
7.1 RMI 概述 155
7.1.1 分布式对象 155
7.1.2 RMI 156
7.1.3 RMI 机制原理 157
7.2 开发 RMI 158
7.2.1 定义远程接口 158
7.2.2 实现远程接口 159
7.2.3 编写服务器类 160
7.2.4 编写客户端 161
7.2.5 部署运行 162
7.2.6 注意事项 164
小结 164
练习 165

第 8 章 国际化 167
8.1 国际化和本地化 169
8.1.1 国际化概述 169
8.1.2 Locale 类 170
8.2 格式化处理 173
8.2.1 数字格式化 173
8.2.2 货币格式化 175
8.2.3 日期格式化 175
8.3 资源包 179
8.3.1 ListResourceBundle 180
8.3.2 PropertyResourceBundle 182
8.4 消息格式化 183
8.5 字符集 186
小结 187
练习 188

第 9 章 正则表达式 189
9.1 正则表达式 191
9.1.1 正则表达式概述 191
9.1.2 模式 191
9.1.3 常用正则表达式 195
9.2 在 Java 中应用正则表达式 196
9.2.1 Pattern 类 197
9.2.2 Matcher 类 198
9.2.3 应用实例 201

小结 205
练习 206

实践篇 207

实践 1　线程 208
实践指导 208
实践 1.G.1 208
实践 1.G.2 211
实践 1.G.3 213
知识拓展 217
1. 定时器 217
2. ThreadLocal 218
拓展练习 220
练习 1.E.1 220
练习 1.E.2 220

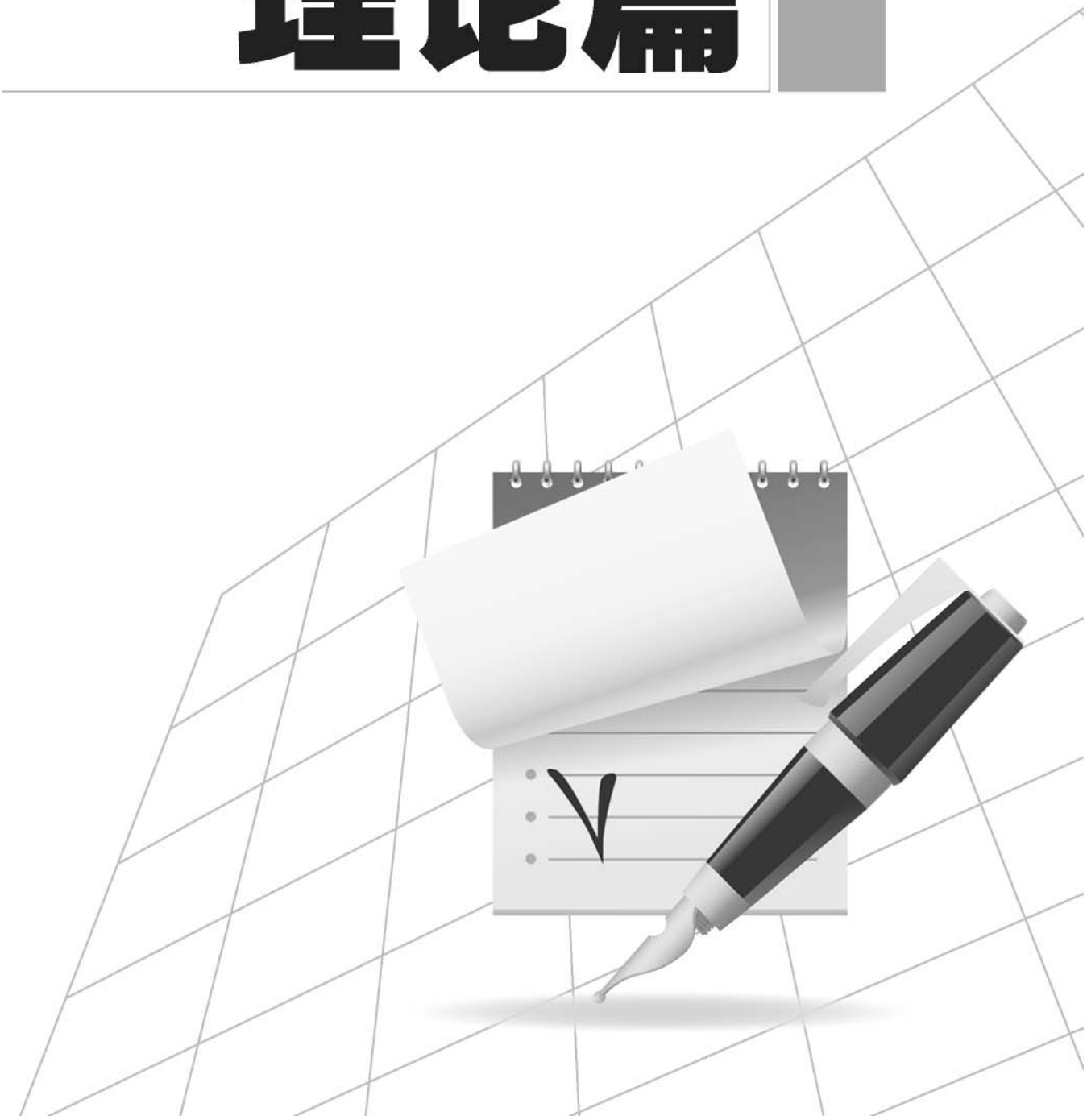
实践 2　网络编程 221
实践指导 221
实践 2.G.1 221
实践 2.G.2 224
实践 2.G.3 227
知识拓展 230
1. 基于 UDP 的网络编程 230
2. 基于 UDP 的组播通信 233
拓展练习 236
练习 2.E.1 236

实践 3　Swing 图形界面（1） 237
实践指导 237
实践 3.G.1 237
实践 3.G.2 238
实践 3.G.3 242
实践 3.G.4 244
知识拓展 246
1. JSplitPane 246
2. JScrollPane 247
拓展练习 249
练习 3.E.1 249
练习 3.E.2 249

实践 4　事件处理 250
实践指导 250
实践 4.G.1 250
实践 4.G.2 253

实践 4.G.3 258
知识拓展 266
1. AdjustmentListener 266
2. 人物眼球转动 268
拓展练习 269
练习 4.E.1 269
练习 4.E.2 269
实践 5　Swing 图形界面（2） 270
实践指导 270
实践 5.G.1 270
实践 5.G.2 273
实践 5.G.3 276
实践 5.G.4 281
知识拓展 291
1. 2D 绘图 291
拓展练习 293
练习 5.E.1 293
实践 6　JDBC 294
实践指导 294
实践 6.G.1 294
实践 6.G.2 296
知识拓展 301
1. 可滚动和可更新的结果集 301
2. 操作 BLOB/CLOB 数据 304
3. 批处理 307
4. 使用 JDBC 连接不同的数据库 309
拓展练习 310
练习 6.E.1 310
练习 6.E.2 310
实践 7　RMI 311
实践指导 311
实践 7.G.1 311
知识拓展 317
1. CORBA 技术 317
2. SOAP 319
拓展练习 320
练习 7.E.1 320
练习 7.E.2 320
附录 A　正则表达式元字符 321

理论篇

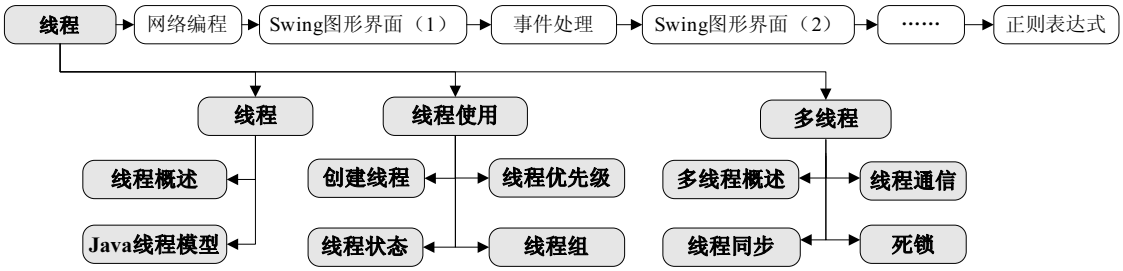


第 1 章　线　　程

本章目标

- 理解线程的基本概念
- 理解 Java 的线程模型
- 掌握 Java 线程的状态和状态转换
- 掌握线程的创建和使用
- 掌握线程优先级的使用
- 掌握线程组的使用
- 理解多线程的概念
- 掌握 Java 的多线程实现
- 掌握线程的同步技巧
- 掌握线程的通信方式
- 理解死锁的概念

学习导航



任务描述

【描述 1.D.1】

通过 Thread 类获取程序的主线程。

【描述 1.D.2】

通过创建 Thread 类的子类，演示线程的创建。

【描述 1.D.3】

通过实现 Runnable 接口，演示线程的创建。

【描述 1.D.4】

演示线程的创建、运行和停止三个状态之间的切换。

【描述 1.D.5】

利用线程的 sleep 状态实现周期性打印信息。

【描述 1.D.6】

利用线程的 interrupt 方法实现线程中断。

【描述 1.D.7】

演示线程优先级的设置及使用。

【描述 1.D.8】

演示线程组的创建和使用。

【描述 1.D.9】

一个简单的多线程例子，能根据用户提供的字符，打印其后连续的 30 个字符。

【描述 1.D.10】

通过多线程，演示不使用同步机制可能出现的情况。

【描述 1.D.11】

通过生产/消费模型，演示线程通信机制的应用。

【描述 1.D.12】

使用资源竞争模型，演示死锁的产生。

1.1 线程基础

线程（Thread）是独立于其他线程运行的程序执行单元。在 Java 体系中，线程在多任务处理中起着举足轻重的作用。

1.1.1 线程概述

线程（轻量级程序）类似于一个程序，也有开始、执行和结束。它是运行在程序内部的一个比进程还要小的单元。使用线程的主要原因在于可以在一个程序中同时运行多个任务。每个 Java 程序都至少有一个线程——主线程。当一个 Java 程序启动时，JVM 会创建主线程，并在该线程中调用程序的 main()方法。

多线程就是同时有多个线程在执行。在多 CPU 的计算机中，多线程的实现是真正的物理上的同时执行，而对于单 CPU 的计算机而言，实现的只是逻辑上的同时执行。在每个时刻，真正执行的只有一个线程，由操作系统进行线程管理调度，但由于 CPU 的速度很快，让人感到像是多个线程在同时执行。

进程是指一种“自包容”的运行程序，有自己的地址空间；线程是进程内部单一的一个顺序控制流。基于进程的特点是允许计算机同时运行两个或更多的程序。基于线程的多任务处理环境中，线程是最小的处理单位。多线程程序在更低的层次中引入多任务处理。

多进程与多线程是多任务的两种类型。多线程与多进程的主要区别在于，线程是一个进程中一段独立的控制流，一个进程可以拥有若干个线程。在多进程设计中各个进程之间的数据块是相互独立的，一般彼此不影响，要通过信号、管道等进行交流。而在多线程设计中，各个线程不一定独立，同一任务中的各个线程共享程序段、数据段等资源。

多线程比多进程更方便于共享资源，而 Java 提供的同步机制解决了线程之间的数据完整性问题，使得多线程设计更易发挥作用。在 Java 程序设计中，动画设计及多媒体应用都会广泛地使用到多线程。

引入线程的优点是：

- 充分利用 CPU 资源。
- 简化编程模型。
- 简化异步事件处理。
- 使 GUI 更有效率。
- 节约成本。

1.1.2 Java 线程模型

Java 的线程模型是面向对象的。在 Java 中建立线程有两种方法：一种是继承 Thread 类；另一种是实现 Runnable 接口，并通过 Thread 和实现 Runnable 的类来建立线程。

Java 通过 Thread 类将线程所必需的功能都封装了起来。要想建立一个线程，必须要有一

个线程执行函数，这个线程执行函数对应 Thread 类的 run()方法；Thread 类还有一个 start()方法，这个方法负责启动线程，当调用 start()方法后，如果线程启动成功，将自动调用 Thread 类的 run()方法。因此，任何继承 Thread 的 Java 类都可以通过 Thread 类的 start()方法来建立线程。如果想运行自己的线程执行函数，那就要重写 Thread 类的 run()方法。

Java 线程模型中还提供了一个标识某个 Java 类是否可作为线程类的接口——Runnable，该接口只有一个抽象方法 run()，也就是 Java 线程模型的线程执行函数。

这两种方法从本质上说是一致的，即都是通过 Thread 类来建立线程，并运行 run()方法的。但它们的区别在于：由于 Java 不支持多继承，因此，这个线程类如果继承了 Thread，就不能再继承其他的类了，而通过实现 Runnable 接口的方法来建立线程，这样的线程类可以在必要的时候继承和业务有关的类，形成清晰的数据模型。

1.2　线程使用

在 Java 中创建线程有几种方法。每个 Java 程序至少包含一个线程：主线程。其他线程都是通过 Thread 构造器或实例化继承类 Thread 的类来创建的。

下述代码用于实现任务描述 1.D.1，通过 Thread 类获取程序的主线程。

【描述 1.D.1】 MainThread.java

```
class MainThread {
    public static void main(String args[]) {
        // 调用 Thread 类的 currentThread()方法获取当前线程
        Thread t = Thread.currentThread();
        System.out.println("主线程是: " + t);
    }
}
```

1.2.1　创建线程

在 Java 中创建线程有两种方法：使用 Thread 类和使用 Runnable 接口。在使用 Runnable 接口时需要建立一个 Thread 实例。因此，无论是通过 Thread 类还是 Runnable 接口建立线程，都必须建立 Thread 类或它的子类的实例。

1. Thread

Java 线程可以通过直接实例化 Thread 对象或实例化继承 Thread 的对象来创建其他线程。Thread 子类需要重写 run 方法，并在 run 方法内部定义线程的功能语句，其常用方法及使用说明见表 1-1。

表 1-1　Thread 类的方法列表

方法名	功能说明
Thread()	构造默认的线程对象
Thread(Runnable target)	使用传递的 Runnable 构造线程对象
Thread(Runnable target,String name)	使用传递的 Runnable 构造名为 name 的线程对象
Thread(ThreadGroup group,Runnable target, String name)	使用传递的 Runnable 在 group 线程组内构造名为 name 的线程对象
final String getName()	返回线程的名称
final boolean isAlive()	如果线程是激活的，则返回 true
final void setName(String name)	将线程的名称设置为由 name 指定的名称
set\getPriority()	设置得到线程优先级
final void join()	等待线程结束
static void sleep(long millis)	用于将线程挂起一段时间，单位毫秒
void start()	调用 run()方法启动线程，开始线程的执行
void stop()	停止线程，已经不建议使用
void interrput()	中断线程
static int activeCount()	返回激活的线程数
static void yield()	使正在执行的线程临时暂停，并允许其他线程执行

下述代码用于实现任务描述 1.D.2，通过创建 Thread 类的子类，演示线程的创建。

【描述 1.D.2】 ThreadDemo.java

```
class Thread1 extends Thread {
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
    }
}
class Thread2 extends Thread {
    public Thread2(String name){
        super(name);
    }
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
    }
}
public class ThreadDemo {
    public static void main(String[] args) {
        Thread1 thread1 = new Thread1();
        // 构造名为 thread2 的线程对象
        Thread2 thread2 = new Thread2("thread2");
```

```
        thread1.start();
        thread2.start();
        // 获取主线程的名字
        System.out.println("["+Thread.currentThread().getName()+"]");
    }
}
```

可能的执行结果：

```
Thread-0
[main]
thread2
```

上述代码定义了两个线程 Thread1 和 Thread2，在创建 thread1 对象时并未指定线程名，因此，所输出的线程名是系统的默认值。而对于输出结果，不仅不同机器之间的结果可能不同，而且在同一机器上多次运行同一程序也可能生成不同结果。另外，线程的执行次序是不定的，除非使用同步机制以强制按特定的顺序执行。

注意　Thread 类的 start 方法不能多次调用，如不能调用两次 thread1.start()方法，否则会抛出一个 IllegalThreadStateException 异常。

2. Runnable

创建线程的另一种方式是实现 Runnable 接口。实现 Runnable 接口的类必须使用 Thread 类的实例才能创建线程。通过 Runnable 接口创建线程的步骤如下：

01 实例化实现 Runnable 接口的类。

02 建立一个 Thread 对象，并将第一步实例化后的对象作为参数传入 Thread 类的构造方法。

03 通过 Thread 类的 start()方法建立线程。

下述代码用于实现任务描述 1.D.3，通过实现 Runnable 接口，演示线程的创建。

【描述 1.D.3】 RunnableDemo.java

```
class Thread3 implements Runnable {
    // 重写 run 方法
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
        for (int i=0;i<30;i++){
            System.out.print("A");
        }
    }
}
```

```
public class RunnableDemo {
    public static void main(String[] args) {
        Thread3 th3 = new Thread3();
        // 将实现 Runnable 的类的实例传入构造函数
        Thread thread = new Thread(th3);
        thread.start();
        // 获取主线程的名字
        System.out.println("["+Thread.currentThread().getName()+"]");
        for (int i=0;i<30;i++){
            System.out.print("C");
        }
    }
}
```

可能的执行结果：

```
Thread-0
AAAAAAAAAAAAAAAAA[main]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCAAAAAAAAAAAA
```

注意 不要直接调用 Thread 类或 Runnable 对象的 run()方法启动，应该通过 Thread 的 start()方法启动线程。

1.2.2 线程状态

线程的状态分为 7 种：born（新生状态）、runnable（就绪状态）、running（运行状态）、waiting（等待状态）、sleeping（睡眠状态）、blocked（阻塞状态）和 dead（死亡状态）。图 1-1 展示了线程的状态和状态之间的转换。

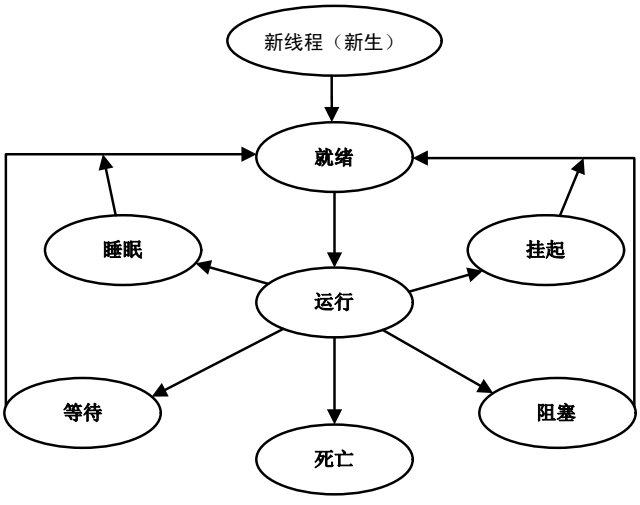


图 1-1 线程状态之间的转换

1. born

当使用 new 来新建一个线程时，一个新的线程就诞生了。除非在构造函数中调用了 start()方法，否则这个新线程将作为一个新对象呆在内存中基本上什么事情也不做。当在方法中对这个线程调用了 start()方法，或者这个线程在程序的其他任何位置使自己的状态由 born 改变为 runnable 之后，调度程序就可以把处理器分配给这个线程。

线程处于等待状态时，可以通过 Thread 类的方法来设置线程的各种属性，如线程的优先级（setPriority）、线程名（setName）和线程的类型（setDaemon）等。

2. runnable、running

把处理器分配给一个处于 runnable 的线程之后，这个线程的状态就变成了 running。如果一个处于 running 状态的线程能够运行到结束或因某个未捕获异常而使线程终止时，它的状态就会变为 dead。否则，这个线程的命运就取决于当前是否还有其他线程等待处理器运行。如果这个线程在当前所有处于 runnable 状态的线程中具有最高优先级，那么它就会继续执行，除非这个线程被运行程序的平台划分为时间片。当线程被划分为时间片时，一个处于 running 状态的线程将分配到一个固定间隔的处理器时间。具有相同优先级的线程的时间片调度将导致这些线程被轮流执行。如果正在执行的线程的代码包含了 yield()方法，那么这个处于 running 状态的线程将停止运行，并把处理器交给其他进程。

可以通过 Thread 类的 isAlive()方法来判断线程是否处于运行状态。当线程处于运行状态时，isAlive()返回 true，当 isAlive()返回 false 时，线程可能处于等待状态，也可能处于停止状态。

下述代码用于实现任务描述 1.D.4，演示线程的创建、运行和停止三个状态之间的切换。

【描述 1.D.4】 LifeThread1.java

```
class LifeThread1 extends Thread {
    public void run() {
        int i = 0;
        while ((++i) < 1000)
            ;
    }
}
public class LifeCycle1 {
    public static void main(String[] args) throws Exception {
        LifeThread1 thread1 = new LifeThread1();
        System.out.println("等待状态[isAlive: " + thread1.isAlive() + "]");
        thread1.start();
        System.out.println("运行状态[isAlive: " + thread1.isAlive() + "]");
        // 等线程 thread1 结束后再继续执行
        thread1.join();
        System.out.println("线程结束[isAlive: " + thread1.isAlive() + "]");
    }
}
```

执行结果如下：

```
等待状态[isAlive: false]
运行状态[isAlive: true]
线程结束[isAlive: false]
```

3. blocked

在线程试图执行某个不能立即完成的任务，并且该线程必须等待其他任务的完成时才能继续，则该线程进入阻塞状态（blocked）。例如，在线程发出输入/输出请求时，这时操作系统将阻塞该线程的执行，直至完成 I/O 操作，操作完成时，该线程便回到就绪状态，可以再次调度该线程，使其执行。

4. sleeping

在线程执行的过程中，可以通过 sleep 方法使线程暂时停止执行，使其进入 sleep 状态。

下述代码用于实现任务描述 1.D.5，利用线程的 sleep 状态实现周期性打印信息。

【描述 1.D.5】 SleepDemo.java

```
class TimeThread extends Thread {
    // 安全退出控制位
    boolean flag = true;
    public void run() {
        while (flag) {
            System.out.println("TimeThread!");
            try {
                // 休眠 1 秒
                Thread.sleep(1000);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
    // 安全退出控制方法
    public void safeStop() {
        flag = false;
    }
}
public class SleepDemo {
    public static void main(String args[]) {
        TimeThread thread = new TimeThread();
        thread.start();
        try {
            // 主线程等待键盘输入
```

```
            System.in.read();
            // 控制 thread 安全控制位，使线程结束
            thread.safeStop();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

上述代码运行期间会每隔 1 秒钟输出“TimeThread!”，直到用户在主线程环境下按下键盘键。

在使用 sleep 方法时有两点需要注意：

- sleep 方法的参数是毫秒。
- 在使用 sleep 方法时必须使用 throws 或放入 try{...}catch{...}语句。

另外，还可以使用 suspend 和 resume 方法方便地挂起和唤醒线程，但这两个方法可能会造成一些不可预料的事情，已被标识为 deprecated，尽量不要使用这两个方法来操作线程。

注意　如果一个线程包含了很长的循环，在循环的每次迭代之后把这个线程切换到 sleep 状态是一种很好的策略，这可以保证其他线程不必等待很长时间才能轮到处理器执行。

5. waiting

如果某个线程的执行条件还未满足，可以调用 wait 方法使其进入等待状态。一旦线程处于等待状态，在另一个线程对其等待对象调用 notify 或 notifyAll 方法时，则线程回到就绪状态。

如果对某个线程调用 interrupt 方法，则该线程会根据线程状态抛出 InterruptedException 异常，对异常进行处理，可以再次调用该线程。

6. dead

线程在下列情况下会结束：

- 线程运行到 run()方法的结尾。
- 线程抛出一个未捕获异常或 Error，或使用 interrupt()方法中断线程。

如在任务描述 1.D.5 中，线程运行期间可以通过调用 safeStop()方法将线程的 flag 控制位设置为 false，结束 while 循环，从而结束线程。

另一种可控的结束线程的方式就是抛出异常或执行 interrupt()方法。

下述代码用于实现任务描述 1.D.6，利用线程的 interrupt 方法实现线程中断。

【描述 1.D.6】 InterruptDemo.java

```
class InterruptThread extends Thread {
    public void run() {
        try {
            System.out.println("在 20 秒之内按任意键中断线程!");
            // 延迟 20 秒
            sleep(2000);
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
public class InterruptDemo {
    public static void main(String[] args) throws Exception {
        Thread thread = new InterruptThread();
        thread.start();
        System.in.read();
        thread.interrupt();
        thread.join();
        System.out.println("线程退出!");
    }
}
```

执行结果如下：

```
在 20 秒之内按任意键中断线程!
sleep interrupted
线程退出!
```

在调用 interrupt()方法后，sleep()方法抛出异常，然后输出错误信息：sleep interrupted。

注意 在 Thread 类中有两个方法可以判断线程是否通过 interrupt 方法被终止。一个是静态的方法 interrupted()，另一个是非静态的方法 isInterrupted()。

1.2.3 线程优先级

线程的优先级代表该线程的重要程度，当有多个线程同时处于可执行状态并等待获得 CPU 时间时，线程调度系统根据各个线程的优先级来决定 CPU 分配时间，优先级高的线程有更大的机会获得 CPU 时间。

每一个线程都有一个优先级。默认情况下优先级通过静态常量 Thread.NORM_PRIORITY 定义，值为 5。每个新线程均继承创建线程的优先级。线程的优先级可以通过 setPriority()/getPriority()方法设置和获取，可将线程的优先级设置为 MIN_PRIORITY（1）和

MAX_PRIORITY（10）之间的值。

下述代码用于实现任务描述 1.D.7，演示线程优先级的设置及使用。

【描述 1.D.7】 PriorityDemo.java

```
class Thread4 extends Thread {
    Thread4(String name) {
        super(name);
    }
    public synchronized void run() {
        System.out.println(this.getName());
        try {
            sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        System.out.println(this.getName() + " 结束!");
    }
}
public class PriorityDemo {
    public static void main(String args[]) throws InterruptedException {
        Thread4 thread1 = new Thread4("first");
        Thread4 thread2 = new Thread4("second");
        Thread4 thread3 = new Thread4("third");
        // 设置优先级为最大,10
        thread1.setPriority(Thread.MAX_PRIORITY);
        thread2.setPriority(Thread.MAX_PRIORITY - 1);
        thread3.setPriority(Thread.MAX_PRIORITY - 2);
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

可能的执行结果：

```
first
second
third
second 结束!
third 结束!
first 结束!
```

当线程睡眠后，优先级就失效了，若想保证优先级继续有效，可考虑使用 join()方法解决。

注意 线程优先级高度依赖于系统。线程优先级不能保证线程的执行次序，应尽量避免使用线程优先级作为构建任务执行顺序的绝对标准。

1.2.4 线程组

某些程序拥有多个线程，如果将它们按照功能归类，可以方便地进行管理。Java 允许创建线程组，实现多线程管理。通过线程组可对其所有线程同时进行操作。例如，通过调用线程组的相应方法来设置其中所有线程的优先级，也可以启动或阻塞其中的所有线程。

Java 的线程组机制的另一个重要作用是线程安全。线程组机制允许通过分组来区分有不同安全特性的线程，对不同组的线程进行不同的处理，还可以通过线程组的分层结构来支持不对等安全措施的采用。

所有线程都隶属于一个线程组，在创建之初，线程被限制到一个系统线程组里，用户也可在创建线程时明确指定组。Java 的 ThreadGroup 类提供了大量的方法，方便对线程组树中的每一个线程组及线程组中的每一个线程进行操作，其常用方法及使用说明见表 1-2。

表 1–2 ThreadGroup 类的方法列表

方法名	功能说明
ThreadGroup(String name)	创建一个新的线程组
int activeCount()	返回线程组中活动线程的数量
int enumerate(Thread []list)	返回所有活动线程的引用
ThreadGroup getParent()	得到这个线程组的父线程组
void interrupt()	中断此线程组和它的子线程组中的所有线程
void list()	打印当前线程组的信息

下述代码用于实现任务描述 1.D.8，演示线程组的创建和使用。

【描述 1.D.8】 ThreadGroupDemo.java

```
class Thread5 extends Thread {
    Thread5(ThreadGroup g, String name) {
        super(g, name);
    }
}
class Thread6 extends Thread5 {
    Thread6(ThreadGroup g, String name) {
        super(g, name);
        // 调用构造函数的同时启动线程
        start();
    }
    public void run() {
        ThreadGroup group = getThreadGroup().getParent().getParent();
        group.list();
```

```
    }
}

public class ThreadGroupDemo {
    public static void main(String[] args) {
        ThreadGroup first = new ThreadGroup("first");
        // 线程组 second 属于线程组 first
        ThreadGroup second = new ThreadGroup(first, "second");
        // 线程组 third 属于线程组 second
        ThreadGroup third = new ThreadGroup(second, "third");
        // 声明线程 one，并使之属于 first 线程组
        Thread one = new Thread5(first, "one");
        // 声明线程 two，并使之属于 third 线程组
        Thread two = new Thread6(third, "two");
    }
}
```

执行结果如下：

```
java.lang.ThreadGroup[name=first,maxpri=10]
   java.lang.ThreadGroup[name=second,maxpri=10]
      java.lang.ThreadGroup[name=third,maxpri=10]
         Thread[two,5,third]
```

1.3　多线程

1.3.1　多线程概述

线程的最主要功能是多任务处理，即多线程。多线程也就是在主线程中有多个线程在运行，多个线程的执行是并发的，在逻辑上“同时”，而不管是否是物理上的“同时”。多线程和传统的单线程在程序设计上最大的区别在于，由于各个线程的控制流彼此独立，使得各个线程之间的代码是乱序执行的，由此带来的线程调度、同步等问题是需要重点留意的。

下述代码用于实现任务描述 1.D.9，是一个简单的多线程例子，能根据用户提供的字符，打印其后连续的 30 个字符。

【描述 1.D.9】 MultiThreadDemo.java

```
public class MultiThreadDemo extends Thread {
    // 存储随机字符
    char c;
    public MultiThreadDemo(String name, char c) {
        super(name);
```

```
        this.c = c;
    }
    public void run() {
        int k;
        char ch = c;
        System.out.println();
        System.out.print(getName() + " : ");
        for (k = 0; k <= 30; k++) {
            ch = (char) (c + k);
            System.out.print(ch + " ");
        }
        System.out.println(getName() + " end!");
    }
    public static void main(String args[]) {
        MultiThreadDemo th1 = new MultiThreadDemo("th1", 'A');
        MultiThreadDemo th2 = new MultiThreadDemo("th2", 'a');
        MultiThreadDemo th3 = new MultiThreadDemo("th3", '0');
        // 启动线程
        th1.start();
        th2.start();
        th3.start();
        // 取得当前活动线程的总数
        System.out.println("activecount = " + Thread.activeCount());
    }
}
```

可能的运行结果：

```
activecount = 4
th1 : A B C D
th2 : a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~   th2 end!
E F G
th3 : 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B H I J K L M N O P Q R S T U V W X Y
Z [ \ ] ^ _ th1 end!
C D E F G H I J K L M N th3 end!
```

上述运算结果在每次运行时都会有所不同，也就是说线程的执行是乱序的，所以在使用多线程时要尤其注意。

1.3.2 线程同步

在 Java 中提供了线程同步的概念以保证某个资源在某一时刻只能由一个线程访问，以保证共享数据及操作的完整性。

Java 使用监控器（也称对象锁）实现同步。每个对象都有一个监控器，使用监控器可以

保证一次只允许一个线程执行对象的同步语句。即在对象的同步语句执行完毕前，其他试图执行当前对象的同步语句的线程都将处于阻塞状态，只有线程在当前对象的同步语句执行完毕后，监控器才会释放对象锁，并让最高优先级的阻塞线程处理它的同步语句。

也可以这样理解对象锁，当拨打公共信息服务台时，接话员（共享对象）可以被多个客户访问（提供服务），但每一次接话员只能为一个客户服务。当其为某个客户服务时，其状态为“忙碌”（获取了对象锁），其他客户只能等待。当接话员为当前的客户服务结束时，其状态就变成“空闲”（释放了对象锁），现在就可以继续为其他客户服务了。

下述代码用于实现任务描述 1.D.10，通过多线程演示不使用同步机制可能出现的情况。

【描述 1.D.10】 NoSyn.java

```
class Share {
    void print(String str) {
        System.out.print("[" + str);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
        System.out.println("]");
    }
}
class Caller implements Runnable {
    String str;
    Share share;
    Thread thread;
    public Caller(Share share, String str) {
        this.share = share;
        this.str = str;
        // 实例化线程
        thread = new Thread(this);
        thread.start();
    }
    public void run() {
        share.print(str);
    }
}
public class NoSyn {
    public static void main(String args[]) throws InterruptedException {
        Share share = new Share();
        Caller call1 = new Caller(share, "A");
        Caller call2 = new Caller(share, "B");
        Caller call3 = new Caller(share, "C");
```

```
    }
}
```

可能的运行结果：

```
[A[B[C]
]
]
```

上述代码中，Share 类为资源类，其实例被 3 个线程共享，并在线程内部调用 Share 类的 print 方法，通过运行结果可以看到，print 方法通过调用 sleep 方法允许执行转换到另一个线程，结果导致 3 个字符串混合输出。

为避免这种情况，须保证其在某一时刻只有一个线程访问 print 方法。在 Java 中使用 synchronized 关键字来实现对象锁。当对象用 synchronized 修饰时，表明该对象在任意时刻只能由一个线程访问。在 Java 中，使用 synchronized 有两种方式：

- 修饰方法，使调用该方法的线程均能获得对象的锁；
- 放在代码块中，修饰对象，当前代码获得对象的锁。

1. 同步方法

实现同步最简单的方式是使用 synchronized 关键字修饰需要同步的方法，此时一旦一个线程进入一个实例的任何同步方法，其他线程将不能进入同一实例的其他同步方法，但该实例的非同步方法仍然能够被调用。

针对任务描述 1.D.10，只需将 synchronized 加到 print 方法前面即可，代码调整如下所示：

```
class Share {
    synchronized void print(String str) {
        System.out.print("[" + str);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
        System.out.println(")");
    }
}
```

此时程序的可能输出结果如下：

```
[A]
[C]
[B]
```

2. 同步块

实现同步另一种有效的做法是使用同步块，只需将对该实例的访问语句放入一个同步块

中即可，形式如下：

```
synchronized(object){
// 需要同步的语句
}
```

使用同步块可以确保只有当前线程能获得被同步对象的对象锁，并可以访问对象的方法。

针对任务描述 1.D.10，使用同步块调整代码，如下所示：

```
class Caller implements Runnable {
    String str;
    Share share;
    Thread thread;
    public Caller(Share share, String str) {
        this.share = share;
        this.str = str;
        // 实例化线程
        thread = new Thread(this);
        thread.start();
    }
    public void run() {
        // 同步 share 对象
        synchronized (share) {
            share.print(str);
        }
    }
}
```

此时程序可能产生与同步方法相同的结果。

注意　synchronized 锁定的是对象，而不是方法或代码块；synchronized 也可以修饰类，当用 synchronized 修饰类时，表示这个类的所有方法都是 synchronized 的。

1.3.3　线程通信

线程同步是无条件地组织其他线程对相关资源的异步访问，这种强大的控制可能带来资源的浪费和工作的不协调。对于“生产/消费”模型，往往会出现“供大于求”或“求大于供”的现象，而此现象的出现就是因为生产者和消费者“沟通”不足引起的。可以使用线程通信使“生产/消费”达到一个较合理的状态，即在没有产品的时候，及时通知生产者，而在生产过剩的情况下及时通知消费者。

在 Java 中可以通过调用 Object 类定义的 wait()、notify()和 notifyAll()方法，使线程之间相互通知事件的发生。要执行这些方法，必须拥有相关对象的锁。

- 调用 wait()方法可以让线程等待，并释放对象锁，直到 interrupt()方法中断它或者另一个线程调用 notify()或 notifyAll()通知它。wait()方法也可以带一个参数，用于指明等待的时间，使用此种方式不需要 notify()或 notifyAll()的唤醒。此方法只能在一个同步方法中调用。
- 调用 notify()方法时可以随机选择一个在该对象中调用 wait()方法的线程，解除它的阻塞。
- 调用 notifyAll()方法可以唤醒等待该对象的所有线程。但唤醒时无法控制唤醒哪个线程，唤醒过程完全由系统来控制。

notify()方法和 notifyAll()方法只能在同步方法或同步块内部使用。

注意 wait()方法区别于 sleep()方法的是：wait()方法调用时会释放对象锁，而 sleep()方法不会。

下述代码用于实现任务描述 1.D.11，通过生产/消费模型演示线程通信机制的应用。

【描述 1.D.11】 WaitDemo.java

```
class Product {
    int n;
    // 为 true 时表示有值可取，为 false 时表示需要放入新值
    boolean valueSet = false;
    // 消费方法
    synchronized void get() {
        // 如果没有值，等待新值放入
        if (!valueSet) {
            try {
                wait();
            } catch (Exception e) {
            }
        }
        System.out.println(Thread.currentThread().getName() + "-Get:" + n);
        // 将 valueSet 设置为 false，表示值已取
        valueSet = false;
        // 通知等待线程，放入新值
        notify();
    }
    // 生产方法
    synchronized void put(int n) {
        // 如果没有值，等待线程取值
        if (valueSet) {
            try {
                wait();
```

```
            } catch (Exception e) {
            }
        }
        this.n = n;
        // 将 valueSet 设置为 true，表示值已放入
        valueSet = true;
        System.out.println(Thread.currentThread().getName() + "-Put:" + n);
        // 通知等待线程，进行取值操作
        notify();
    }
}
class Producer implements Runnable {
    Product product;
    Producer(Product product) {
        this.product = product;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int k = 0;
        // 生产 5 次
        for (int i = 0; i < 5; i++) {
            product.put(k++);
        }
    }
}
class Consumer implements Runnable {
    Product product;
    Consumer(Product product) {
        this.product = product;
        new Thread(this, "Consumer").start();
    }
    public void run() {
        // 消费 5 次
        for (int i = 0; i < 5; i++) {
            product.get();
        }
    }
}
public class WaitDemo {
    public static void main(String args[]) {
        // 共享的生产/消费实例
        Product product = new Product();
        // 指定生产线程
```

```
        Producer producer = new Producer(product);
        // 指定消费线程
        Consumer consumer = new Consumer(product);
    }
}
```

执行结果如下：

```
Producer-Put:0
Consumer-Get:0
Producer-Put:1
Consumer-Get:1
Producer-Put:2
Consumer-Get:2
Producer-Put:3
Consumer-Get:3
Producer-Put:4
Consumer-Get:4
```

上述代码描述了典型的生产/消费模型，其中 Product 类是资源类，用于为生产者和消费者提供资源；Producer 是生产者，产生队列输入；Consumer 是消费者，从队列中取值。

定义 Product 类时，使用 synchronized 修饰 put()和 get()方法，确保当前实例在某一时刻只有一种状态：要么生产，要么消费；在 put()和 get()方法内部，通过信号量 valueSet 的取值，利用 wait()和 notify()方法的配合实现线程间的通信，确保生产和消费的相互依赖关系。

main()方法中，通过声明 Product 类的实例，并将声明的实例传入生产线程和消费线程，使两个线程在产生“资源竞争”的情况下，保持良好的生产消费关系。

1.3.4 死锁

通过对象锁，可以很方便地实现数据同步，但对象锁引起的等候很容易导致另外一种问题——“死锁”。

所谓死锁，是指两个或多个线程都在等待对方释放对象资源而进入的一种不可“调节”的状态。在程序设计中，死锁是无法测知或避开的（就像不知道操作系统什么时候死机一样），尽管这种情况并非经常出现，但一旦碰到，程序的调试将变得异常艰难。

最常见的死锁模式是：当线程 A 拥有 obj1 上的对象锁时，为完成任务同时需要获取 obj2 上的对象锁，而此时线程 B 正拥有 obj2 的对象锁，其为完成任务需要获取 obj1 上的对象锁。此时两个线程在对象资源上既无法获取（对方线程未执行完毕）也无法释放（自身线程未执行完毕），程序进入无限期的等待，死锁就发生了。

下述代码用于实现任务描述 1.D.12，使用资源竞争模型演示死锁的产生。

【描述 1.D.12】 DeadLockDemo.java

```
public class DeadLockDemo implements Runnable {
    private boolean flag;
    // 使用 obj1 和 obj2 模拟两个对象资源
    static Object obj1 = new Object();
    static Object obj2 = new Object();
    public void run() {
        String name = Thread.currentThread().getName();
        System.out.println(name + " flag : " + flag);
        /*
         * 如果 flag 为 true，首先获取 obj1 资源，等待 500 毫秒后，获取 obj2 资源
         * 如果 flag 为 false，首先获取 obj2 资源，等待 500 毫秒后，获取 obj1 资源
         */
        if (flag) {
            synchronized (obj1) {
                try {
                    System.out.println(name + " 休眠 500 毫秒 ,等待 obj2 ");
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized (obj2) {
                    System.out.println("获取 obj2 ");
                }
            }
        } else {
            synchronized (obj2) {
                try {
                    System.out.println(name + " 休眠 500 毫秒 ,等待 obj1 ");
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized (obj1) {
                    System.out.println("获取 obj1 ");
                }
            }
        }
    }
    public static void main(String[] args) {
        DeadLockDemo obj1 = new DeadLockDemo();
        DeadLockDemo obj2 = new DeadLockDemo();
```

```
        /*
         * 构造资源竞争模型
         */
        // 将 flag 设置为 true，为执行 if 语句
        obj1.flag = true;
        // 将 flag 设置为 false，为执行 else 语句
        obj2.flag = false;
        Thread thread1 = new Thread(obj1);
        Thread thread2 = new Thread(obj2);
        thread1.start();
        thread2.start();
    }
}
```

执行结果如下：

```
Thread-0 flag : true
Thread-0 休眠 500 毫秒 ,等待 obj2
Thread-1 flag : false
Thread-1 休眠 500 毫秒 ,等待 obj1
// 进入无休止的等待
```

上述代码中线程 thread1 在获取了 obj1 资源后等待 500 毫秒，几乎同时（计算机执行速度快）thread2 在获取了 obj2 资源后等待 500 毫秒，当睡眠时间到达后，thread1 为继续执行需要获得 obj2 的对象锁，而与此同时 thread2 正拥有 obj2 的对象锁，thread2 也为执行需要等待 obj1 对象锁的释放，thread1 和 thread2 进入“僵持”阶段。当死锁产生后，除非中断（牺牲）某个线程，否则死锁将不可排解。

注意 死锁是多线程编程的产物，我们无法彻底避免，只能尽可能预防。譬如在获取多个对象锁时，尽量在所有线程中都以相同的顺序获取，不要滥用同步机制，避免无谓的同步控制等。

小结

通过本章的学习，学生应该能够学会：

- 线程（Thread）是独立于其他线程运行的程序执行单元。
- 线程的主要应用在于可以在一个程序中同时运行多个任务。
- 通过继承 Thread 类或实现 Runnable 接口创建线程类。
- Java 线程有新生、就绪、运行、睡眠、等待、阻塞、死亡 7 个状态。
- 通过设置线程的优先级控制线程的执行次序。
- Java 程序启动时的入口是 main()函数，通常称为程序的主线程。

- Java 引用“监视器”的概念实现线程同步。
- 通过同步块和同步方法两种方式来实现同步。
- Java 线程之间可以通过 wait()、notify()和 notifyAll()方法实现通信。
- 线程同步可能导致死锁的产生。

练习

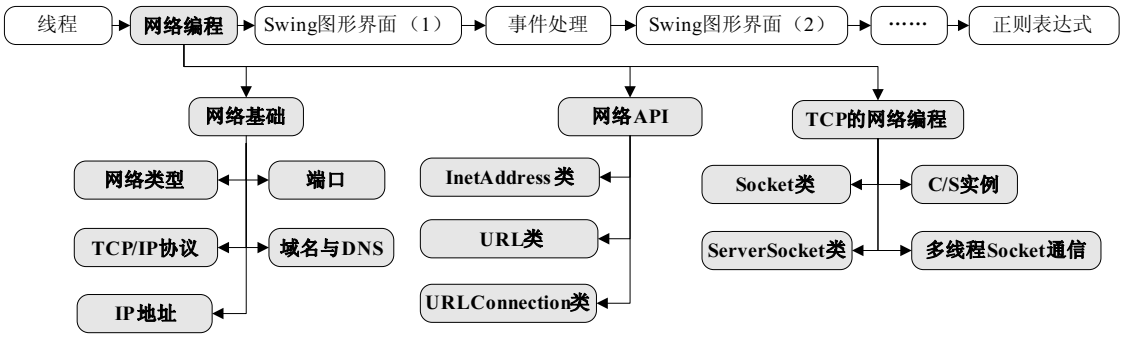
1. 下面_____是线程类。
 A. Runnable　　B. Thread　　C. ThreadGroup　　D. Throwable
2. 要建立一个线程，可以从下面_____接口继承。
 A. Runnable　　B. Thread　　C. Run　　D. Throwable
3. 下面让线程休眠 1 分钟的正确方法是_____。
 A. sleep(1)　　B. sleep(60)　　C. sleep(1000)　　D. sleep(60000)
4. 列举让线程处于不运行状态的方法_____。
5. 线程同步的关键字为_____。
6. 简述什么是多线程。
7. 使用线程实现能根据用户传入的时间间隔打印当前时间，时间格式为“年-月-日 时：分：秒”。
8. 用代码来模拟铁路售票系统，实现通过四个售票点发售某日某次列车的 100 张车票，一个售票点用一个线程表示。

第 2 章　网络编程

本章目标

- 理解计算机网络编程的概念
- 理解 TCP/IP 协议规范
- 理解域名与 DNS 的概念
- 掌握基于 URL 的网络编程
- 掌握基于 TCP 的 C/S 网络编程
- 掌握基于 Socket 的低层次 Java 网络编程
- 掌握基于 Socket 的多线程通信

学习导航



任务描述

【描述 2.D.1】

通过 InetAddress 类获取本机的地址信息和指定域名的地址信息。

【描述 2.D.2】

根据指定的路径构造 URL 对象，并获取当前 URL 对象的相关属性。

【描述 2.D.3】

根据指定的路径构造 URL 对象，并从当前 URL 对象中读取相关数据。

【描述 2.D.4】

使用 URLConnection 类从 Web 服务器上读取文件的信息并打印。

【描述 2.D.5】

基于 Socket 通信，实现单客户端和服务器交互的 C/S 结构。

【描述 2.D.6】

基于线程，实现多客户端和服务器的 Socket 通信。

2.1 网络基础

随着计算机技术的发展，基于网络的应用已成为时代发展的重要标志。通过网络可以将分布在不同区域的计算机、外设、数据资源等用通信线路互联成一个规模大、功能强、资源共享的网络系统，使众多的终端可以方便地互相传递信息，共享硬件、软件和数据信息等资源。

2.1.1 网络类型

计算机网络，按照不同的分类方式来划分，可以分为不同的类型。

按照网络的地理位置可以分为：

- **局域网（LAN）**：一般限定在较小的区域内，小于 10km 的范围，通常采用有线的方式连接起来。
- **城域网（MAN）**：规模局限在一座城市的范围内，10~100km 的区域。
- **广域网（WAN）**：网络跨越国界、洲界，甚至全球范围。

目前局域网和广域网是网络的热点。局域网是组成其他两种类型网络的基础，城域网一般都加入了广域网。广域网的典型代表是 Internet。

按照服务方式可以分为：

- **客户机/服务器网络（Client/Server）**：服务器是指专门提供服务的高性能计算机或专用设备，客户机是用户计算机。此种服务方式是客户机向服务器发出请求并获得服务的一种网络形式，多台客户机可以共享服务器提供的各种资源。客户机/服务器是最常用、最重要的一种网络类型，网络安全性容易得到保证，计算机的权限、优先级易于控制，监控容易实现，网络管理能够规范化。
- **对等网（Peer-to-Peer）**：对等网不要求文件服务器，每台客户机都可以与其他每台客户机对话，共享彼此的信息资源和硬件资源，组网的计算机一般类型相同。这种网络方式灵活方便，但是较难实现集中管理与监控，安全性较低，较适合于部门内部协同工作的小型网络。

目前较为流行的网络编程模型是客户机/服务器（C/S）结构，即通信双方一方作为服务器等待客户提出请求并予以响应，客户则在需要服务时向服务器提出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也能及时得到服务。

2.1.2 TCP/IP 协议

通信协议即网络中（包括互联网）传递、管理信息的一些规范。如同人与人之间相互交流是需要遵循一定的语言定义一样，计算机之间的相互通信也需要共同遵守一定的规则，这

些规则就称为网络协议。常见的协议有：TCP/IP 协议、IPX/SPX 协议、NetBEUI 协议等。IPX/SPX 一般用于局域网中。如果访问 Internet，则必须在网络协议中添加 TCP/IP 协议。

TCP/IP 是“Transmission Control Protocol/Internet Protocol”的简写，中文译名为传输控制协议/网际协议。TCP/IP 是一种网络通信协议，它规范了网络上的所有通信设备，尤其是一个主机与另一个主机之间的数据往来格式及传送方式。TCP/IP 是 Internet 的基础协议，也是一种计算机数据打包和寻址的标准方法。在数据传送中，可以形象地理解为有两个信封，TCP 和 IP 就像是信封，要传递的信息被划分成若干段，每一段塞入一个 TCP 信封，并在该信封面上记录有分段号的信息，再将 TCP 信封塞入 IP 大信封，发送上网。在接收端，TCP 软件包收集信封，抽出数据，按发送前的顺序还原，并加以校验，若发现差错，TCP 将会要求重发。因此，TCP/IP 在 Internet 中几乎可以无差错地传送数据。对普通用户来说，并不需要了解网络协议的整个结构，仅需了解 IP 的地址格式，即可与世界各地进行网络通信。

1. IP 协议

IP 是 TCP/IP 的关键部分，也是网络层中最重要的协议。IP 协议提供了能适应各种各样网络硬件的灵活性，对底层网络硬件几乎没有任何要求，任何网络只要可以从一个地点向另一个地点传送二进制数据，就可以使用 IP 协议加入 Internet。

如果希望能在 Internet 上进行交流和通信，则每台连上 Internet 的计算机都必须遵守 IP 协议。为此使用 Internet 的每台计算机都必须运行 IP 软件，以便时刻准备发送或接收信息。

IP 协议对于网络通信有着重要的意义：网络中的计算机通过安装 IP 软件，使许许多多的局域网络构成了一个庞大而又严密的通信系统。从而使 Internet 看起来好像是真实存在的，但实际上它是一种并不存在的虚拟网络，只不过是利用 IP 协议把全世界上所有愿意接入 Internet 的计算机局域网络连接起来，使得它们彼此之间都能够通信。

IP 协议是一个无连接、不可靠的协议。在向另一台计算机传输数据之前，它不交换控制信息，数据包只是传送到目的主机，并且假设能被正确处理。由于 IP 协议并不重新传输已丢失的数据包或检测受损害的数据，所以 IP 协议是不可靠的。这种功能可以通过 TCP 来实现。

2. TCP 协议

尽管通过安装 IP 软件可以保证计算机之间发送和接收数据，但 IP 协议还不能解决数据分组在传输过程中可能出现的问题。因此，若要解决可能出现的问题，连上 Internet 的计算机还需要安装 TCP 协议来提供可靠的并且无差错的通信服务。

TCP 协议被称做一种端对端协议。它为两台计算机之间的连接起了重要作用：当一台计算机需要与另一台远程计算机连接时，TCP 协议会让它们建立一个连接，发送、接收数据及终止连接。Internet 是一个庞大的国际性网络，网络上的拥挤和空闲时间总是交替不定的，在网络上的传送距离也远近不同，传输数据所用时间也会变化不定。TCP 协议利用重发技术和拥塞控制机制，向应用程序提供可靠的通信连接，使它能够自动适应网上的各种变化。即使在 Internet 暂时出现堵塞的情况下，TCP 也能够保证通信的可靠；TCP 协议具有自动调整“超时值”的功能，能很好地适应 Internet 上各种各样的变化，确保传输数值的正确。

从上述描述可以得知：IP 协议只保证计算机能发送和接收分组数据，而 TCP 协议则可

提供一个可靠的、可流控的、全双工的信息流传输服务。

综上所述，虽然 IP 和 TCP 这两个协议的功能不尽相同，也可以分开单独使用，但它们是在同一时期作为一个协议来设计的，并且在功能上也是互补的，只有两者的结合，才能保证 Internet 在复杂的环境下正常运行。凡是要连接到 Internet 的计算机，都必须同时安装和使用这两个协议，因此在实际中常把这两个协议统称为 TCP/IP 协议。

2.1.3 IP 地址

为了实现各主机间的通信，每台主机都必须有一个唯一的网络地址（Internet 的网络地址是指连入 Internet 网络的计算机的地址编号），通过网络地址唯一地标识一台计算机，这个地址就叫做 IP（Internet Protocol）地址，即用网际协议表示的地址。

IP 地址是一个 32 位的二进制地址，为了便于记忆，将它们分为 4 组，每组 8 位，由小数点分开，用 4 字节来表示。用点分开的每个字节的数值范围是 0~255，如 202.116.0.1，此种书写方法叫做点分表示法。

通过 IP 地址可以确认网络中的任何一个网络和计算机，而要识别其他网络或其中的计算机，则是根据这些 IP 地址的分类来确定的。一般将 IP 地址按节点计算机所在网络规模的大小分为 A、B、C 三类，每个类别的网络标识和主机标识各有规则。

1. A 类地址

A 类地址的表示范围为：0.0.0.0~126.255.255.255，默认网络掩码为：255.0.0.0。

A 类地址分配给规模特别大的网络使用。A 类网络用第一组数字表示网络本身的地址，后面三组数字作为连接于网络上的主机地址。A 类地址分配给具有大量主机（直接个人用户）而局域网络个数较少的大型网络，例如，IBM 公司的网络。

2. B 类地址

B 类地址的表示范围为：128.0.0.0~191.255.255.255，默认网络掩码为：255.255.0.0。

B 类地址分配给一般的中型网络。B 类网络用第一和第二组数字表示网络的地址，后面两组数字代表网络上的主机地址。

3. C 类地址

C 类地址的表示范围为：192.0.0.0~223.255.255.255，默认网络掩码为：255.255.255.0。

C 类地址分配给小型网络，如一般的局域网和校园网，它可连接的主机数量是最少的，采用把所属的用户分为若干的网段进行管理。C 类网络用前三组数字表示网络的地址，最后一组数字作为网络上的主机地址。

实际上，还存在着 D 类地址和 E 类地址。但这两类地址用途比较特殊：D 类地址称为广播地址，供特殊协议向选定的节点发送信息时用；E 类地址保留给将来使用。

2.1.4　端口

端口（Port），可以认为是计算机与外界通信交流的出口。其中硬件领域的端口又称接口，如：USB 端口、串行端口等。软件领域的端口一般指网络中面向连接服务和无连接服务的通信协议端口，是一种抽象的软件结构，包括一些数据结构和 I/O（基本输入输出）缓冲区。在这里，特指软件领域中的端口。

可以这样理解 TCP/IP 协议中的端口：如果把 IP 地址比做一间房子，端口就是出入这间房子的门，一个 IP 地址的端口可以有 65 536（即 256×256）个。端口是通过端口号来标记的，端口号只有整数，范围是 0~65 535。

按端口号的范围可以分为三大类：

- 公认端口（Well Known Port）：从 0 到 1023，它们紧密绑定（binding）于一些服务。通常这些端口的通信明确表明了某种服务的协议。例如：80 端口实际上总是 HTTP 通信。
- 注册端口（Registered Port）：从 1024 到 49 151，它们松散地绑定于一些服务。也就是说有许多服务绑定于这些端口，这些端口同样用于许多其他目的。例如：8080 是 Tomcat 的默认服务端口。
- 动态和/或私有端口（Dynamic and/or Private Port）：从 49 152 到 65 535。理论上，不应为服务分配这些端口。

表 2-1 列举了一些常用端口及其对应的服务。

表 2-1　常用端口及其对应的服务

端口号	服务
7	Echo 服务端口
21	FTP 服务端口
23	Telnet 服务端口
25	SMTP 服务端口
80	HTTP 服务端口

注意　在建立自己的网络服务时，尽量避免使用公认端口的端口值。

2.1.5　域名与 DNS

1. 域名

由于 IP 地址是数字标识，使用时难以记忆和书写，因此在 IP 地址的基础上又发展出一种符号化的地址方案，来代替数字型的 IP 地址。每一个符号化的地址都与特定的 IP 地址对应，如 www.baidu.com，这样网络上的资源访问起来就容易得多了。这个与网络上的数字型 IP 地址相对应的字符型地址，就被称为域名。

域名（Domain Name）是由一串用点分隔的名字组成的 Internet 上某一台计算机或计算

机组的名称，用于在数据传输时标识计算机的电子方位（有时也指地理位置）。域名一般由若干个 a~z 的 26 个字母、0~9 的 10 个阿拉伯数字及“-”、“.”符号构成，并按一定的层次和逻辑排列（目前也有一些国家在开发其他语言的域名，如中文域名）。域名不仅便于记忆，而且即使在 IP 地址发生变化的情况下，通过改变解析对应关系，域名仍可保持不变。

按照级别，常用的域名分为：

- **国际域名：**也叫国际顶级域名，这也是使用最早、最广泛的域名。例如，表示工商企业的.com，表示网络提供商的.net，表示非营利组织的.org 等。
- **国内域名：**也叫国内顶级域名。即按照国家（或地区）的不同分配不同后缀，这些域名即该国（或地区）的国内顶级域名。目前二百多个国家和地区都按照 ISO3166 国家代码分配了顶级域名，如中国是 cn，美国是 us，日本是 jp 等。
- **二级域名：**指顶级域名之下的域名。在国际顶级域名下，它是指域名注册人的网上名称，如 ibm、yahoo、microsoft 等；在国家顶级域名下，它是表示注册企业类别的符号，如 com、edu、gov、net 等。
- **三级域名：**一般应用于中小企业及个人注册使用。

在实际使用和功能上，国际域名与国内域名没有任何区别，都是 Internet 上的具有唯一性的标识。如 baidu（www.baidu.com）网址是由三部分组成的，标号“baidu”是这个域名的主体，而最后的标号“com”则是该域名的后缀，代表的这是一个 com 国际域名，是顶级域名，而前面的 www 是网络名，baidu.com 为 www 的域名。

2. DNS

在 Internet 上，域名与 IP 地址之间是一对一（或多对一）的，域名虽然便于人们记忆，但机器之间只能互相认识 IP 地址，它们之间的转换工作称为域名解析。域名解析需要由专门的域名解析服务器来完成，DNS（Domain Name Server）就是进行域名解析的服务器。通过 DNS 服务可以将用户输入的域名解析为与之相关的 IP 地址，从而唯一确定该域名所绑定的域层次结构中的计算机和网络服务。

2.2 网络 API

Java 最初是作为一种网络编程语言出现的，它能够使用网络上的各种资源和数据与服务器建立各种传输通道，实现数据的传输。

Java 中有关网络方面的功能都定义在 java.net 程序包中。

2.2.1 InetAddress 类

地址是网络通信的基础。在 Java 中使用 InetAddress 类来封装前面讨论的 IP 地址和该地址的域名。InetAddress 类内部隐藏了地址数字，它不需要用户了解如何实现地址的细节。

InetAddress 类无构造方法，不能直接创建其对象，但可以通过该类的静态方法创建一个 InetAddress 对象或 InetAddress 数组，其常用方法及功能见表 2-2。

表 2–2　InetAddress 类的方法列表

方法名	功能说明
public static InetAddress getLocalHost()	获得本地机的 InetAddress 对象，当查找不到本地机器的地址时，抛出一个 UnknownHostException 异常
public static InetAddress getByName (String host)	获得由 host 指定的 InetAddress 对象，host 是计算机的域名（或 IP）。如果找不到主机会抛出 UnknownHostException 异常
public static InetAddress[] getAllByName(String host)	获得具有相同名字的一组 InetAddress 对象，如果找不到主机会抛出 UnknownHostException 异常
public static InetAddress getByAddress(byte[] addr)	获取 addr 所封装的 IP 地址对应的 InetAddress 对象，如果找不到主机会抛出 UnknownHostException 异常
public String getCanonicalHostName()	从域名服务中获得标准的主机名
public bytes[] getHostAddress()	获得主机 IP 地址
public String getHostName()	获得主机名
public String toString()	获得主机名和 IP 地址的字符串

下述代码用于实现任务描述 2.D.1，通过 InetAddress 类获取本机的地址信息和指定域名的地址信息。

【描述 2.D.1】 InetAddressDemo.java

```
public class InetAddressDemo {
    public static void main(String para[]) throws UnknownHostException {
        // 获取本机地址信息
        InetAddress IP0 = InetAddress.getLocalHost();
        System.out.println("IP0.getCanonicalHostName()= "
                + IP0.getCanonicalHostName());
        System.out.println("IP0.getHostAddress()= " + IP0.getHostAddress());
        System.out.println("IP0.getHostName()= " + IP0.getHostName());
        System.out.println("IP0.toString()= " + IP0.toString());
        System.out.println("====================================");
        // 获取指定域名地址信息
        InetAddress IP1 = InetAddress.getByName("www.baidu.com");
        System.out.println("IP1.getCanonicalHostName()= "
                + IP1.getCanonicalHostName());
        System.out.println("IP1.getHostAddress()= " + IP1.getHostAddress());
        System.out.println("IP1.getHostName()= " + IP1.getHostName());
        System.out.println("IP1.toString()= " + IP1.toString());
        System.out.println("====================================");
        // 比较两个 InetAddress
        InetAddress IP2 = InetAddress.getByAddress(IP1.getAddress());
```

```
        System.out.println("IP2.getCanonicalHostName()= "
                + IP2.getCanonicalHostName());
        System.out.println("IP2.getHostAddress()= " + IP2.getHostAddress());
        System.out.println("IP2.getHostName()= " + IP2.getHostName());
        if (IP2.equals(IP1))
            System.out.println("IP2 equals IP1");
        else
            System.out.println("IP2 not equals IP1");
    }
}
```

执行结果如下：

```
IP0.getCanonicalHostName()= dingcq
IP0.getHostAddress()= 10.10.9.113
IP0.getHostName()= dingcq
IP0.toString()= dingcq/10.10.9.113
====================================
IP1.getCanonicalHostName()= xd-22-5-a8.bta.net.cn
IP1.getHostAddress()= 202.108.22.5
IP1.getHostName()= www.baidu.com
IP1.toString()= www.baidu.com/202.108.22.5
====================================
IP2.getCanonicalHostName()= xd-22-5-a8.bta.net.cn
IP2.getHostAddress()= 202.108.22.5
IP2.getHostName()= xd-22-5-a8.bta.net.cn
IP2 equals IP1
```

注意 在需要获得 Internet 上的域名所对应的地址信息时，需保证运行环境能访问 Internet，否则将抛出 UnknownHostException 异常。

2.2.2 URL 类

URL 是 Uniform Resource Locator（统一资源定位器）的缩写，它表示 Internet 上某一资源的地址。通过 URL 可以访问 Internet 上的各种网络资源，如最常见的 WWW、FTP 站点。浏览器通过解析给定的 URL 可以在网络上查找相应的文件或其他资源。

URL 是最为直观的一种网络定位方法，符合人们的语言习惯，容易记忆。为了处理方便，Java 将 URL 封装成 URL 类，可以通过 URL 对象记录下完整的 URL 信息。

http://www.myweb.com:8080/index.htm 是一个合法的 URL 示例。一个完整的 URL 由协议名、主机名（主机 IP）、端口号和文件路径四部分组成：

- **协议名（protocol）：**指明获取资源所使用的传输协议，如 http、ftp 等，使用冒号（:）来将它与其他部分相隔离，如前面示例中的 http。

- **主机名（host）：**指定获取资源的域名，此部分由左边的双斜线（//）和右边的单斜线（/）或可选冒号（:）限制，如前面示例中的 www.myweb.com。
- **端口（port）：**指定服务的端口号，是可选的参数，由主机名左边的冒号（:）和右边的斜线（/）限制，如前面示例中的 8080。
- **文件路径（file）：**指定访问的文件名及路径，如前面示例中的 index.htm。

URL 类常用方法及功能见表 2-3。

表 2–3　URL 类的方法列表

方法名	功能说明
public URL(String spec)	构造方法，根据指定的 spec 来创建一个 URL 对象
public URL(String protocol,String host,int port,String file)	构造方法，根据指定的协议、主机名、端口号、文件路径及文件名创建一个 URL 对象
public URL(String protocol, String host, String file)	构造方法，根据指定的协议、主机名、路径及文件名创建 URL 对象
public String getProtocol()	获取该 URL 的协议名
public String getHost()	获取该 URL 的主机名
public int getPort()	获取该 URL 的端口号，如果没有设置端口，返回-1
public String getFile()	获取该 URL 的文件名
public String getQuery()	获取该 URL 的查询信息
public String getPath()	获取该 URL 的路径
public String getRef()	获得该 URL 的引用

注意　类 URL 的构造方法都声明抛出异常 MalformedURLException，因此构造 URL 对象时，需要对此异常进行处理。

下述代码用于实现任务描述 2.D.2，根据指定的路径构造 URL 对象，并获取当前 URL 对象的相关属性。

【描述 2.D.2】 URLDemo.java

```
public class URLDemo {
    public static void main(String[] args) throws Exception {
        URL Aurl = new URL("http://java.sun.com/docs/books/");
        URL tuto = new URL(Aurl, "tutorial.intro.html#DOWNLOADING");
        System.out.println("protocol=" + tuto.getProtocol());
        System.out.println("host =" + tuto.getHost());
        System.out.println("filename=" + tuto.getFile());
        System.out.println("port=" + tuto.getPort());
        System.out.println("ref=" + tuto.getRef());
        System.out.println("query=" + tuto.getQuery());
        System.out.println("path=" + tuto.getPath());
```

```
    }
}
```

执行结果如下：

```
protocol=http
host=java.sun.com
filename=/docs/books/tutorial.intro.html
port=-1
ref=DOWNLOADING
query=null
path=/docs/books/tutorial.intro.html
```

当得到一个 URL 对象后，可以通过调用 URL 的 openStream()方法读取指定的 WWW 资源。方法 openSteam()与指定的 URL 建立连接并返回 InputStream 类的对象以从这一连接中读取数据。

下述代码用于实现任务描述 2.D.3，根据指定的路径构造 URL 对象，并从当前 URL 对象中读取相关数据。

【描述 2.D.3】 URLReader.java

```
public class URLReader {
    public static void main(String[] args) throws Exception {
        // 构建一个 URL 对象
        URL baidu = new URL("http://www.baidu.com/");
        // 获取输入流，构造一个 BufferedReader 对象
        BufferedReader br = new BufferedReader(new InputStreamReader(baidu
                .openStream()));
        String inputLine;
        // 循环读取并打印数据
        while ((inputLine = br.readLine()) != null)
            System.out.println(inputLine);
        // 关闭输入流
        br.close();
    }
}
```

执行结果将是指定网址资源对应文件的源码。

2.2.3 URLConnection 类

通过 URL 的方法 openStream()，只能从网络上读取数据，如果需要输出数据，此时就要用到 URLConnection 类。URLConnection 是一个抽象类，代表与 URL 指定的数据源的动态连接，URLConnection 类提供比 URL 类更强的服务器交互控制，其允许用 POST 或 PUT 和其他 HTTP 请求方法将数据送回服务器，其常用方法及功能见表 2-4。

表 2-4　URLConnection 类的常用方法及其功能说明

方法名	功能说明
public int getContentLength()	获得文件的长度
public String getContentType()	获得文件的类型
public long getDate()	获得文件创建的时间
public long getLastModified()	获得文件最后修改的时间
public InputStream getInputStream()	获得输入流，以便读取文件的数据
public OutputStream getOutputStream()	获得输出流，以便输出数据

下述代码用于实现任务描述 2.D.4，使用 URLConnection 类从 Web 服务器上读取文件的信息，并打印文件的信息。

【描述 2.D.4】 URLConnectionDemo.java

```
public class URLConnectionDemo {
    public static void main(String[] args) throws Exception {
        // 构建一个 URL 对象
        URL baidu = new URL("http://www.baidu.com/");
        //由 URL 对象获取 URLConnection 对象
        URLConnection uc=baidu.openConnection();
        //由 URLConnection 获取输入流，并构造 BufferedReader 对象
        BufferedReader br =
        new BufferedReader(new InputStreamReader(uc.getInputStream()));
        String inputLine;
        // 循环读取并打印数据
        while ((inputLine = br.readLine()) != null)
            System.out.println(inputLine);
        // 关闭输入流
        br.close();
    }
}
```

执行结果与 2.D.3 一致。实际上，URL 类的方法 openSteam()是通过 URLConnection 来实现的，它等价于 openConnection().getInputStream()。

2.3　基于 TCP 的网络编程

套接字（Socket）允许程序将网络连接当成一个流，可以向这个流中写字节，也可以从这个流中读取字节。套接字为程序员屏蔽了网络的底层细节，如媒体类型、信息包的大小、网络地址、信息的重发等。

TCP/IP 套接字用于在主机和 Internet 之间建立可靠的、双向的、持续的、点对点的流式

连接。一个套接字可以用来建立 Java 的输入/输出系统到其他的驻留在本地机或 Internet 上的任何机器的程序的连接。

Java 中有两种 TCP 套接字：一种是服务器套接字（ServerSocket）；另一种是客户端套接字（Socket）。ServerSocket 类设计成在等待客户建立连接之前不做任何事的“监听器”。Socket 类为建立连向服务器套接字及启动协议交换而设计。利用 Socket 类的方法，就可以实现两台计算机之间的通信。

2.3.1 Socket 类

Socket 类是网络上运行的两个程序间双向通信的一端，它既可以接受请求，也可以发送请求，利用它可以较为方便地编写网络上数据的传递方法，其常用方法及功能见表 2-5。

表 2-5 Socket 类的方法列表

方法名	功能说明
public Socket(String host, int port)	创建一个到主机 host、端口号为 port 的套接字，并连接到远程主机
public Socket (InetAddress host, int port)	创建一个套接字，使用 host 中封装的主机信息，端口号为 port，并连接到主机
public InetAddress getInetAddress()	返回连接到远程主机的地址，如果连接失败，则返回以前连接的主机
public int getPort()	返回 Socket 连接到远程主机的端口号
public int getLocalPort()	返回本地连接终端的端口号
public InputStream getInputStream()	返回一个输入流，利用这个流就可以从套接字读取数据
public OutputStream getOutputStream()	返回一个输出流，可以在应用程序中写数据到套接字的另一端
public synchronized void close()	关闭当前 Socket 连接

注意 在创建 Socket 时如果发生错误，将产生 IOException 异常，在程序中必须对之作出处理。所以在创建 Socket 时必须捕获或抛出异常。另外，在选择端口号时，最好选择一个大于 1023 的数以防止发生冲突。

一般情况下 Socket 的工作步骤如下：

01 根据指定地址和端口创建一个 Socket 对象。
02 调用 getInputStream()方法或 getOutputStream()方法打开连接到 Socket 的输入/输出流。
03 客户端与服务器根据一定的协议交互，直到关闭连接。
04 关闭客户端的 Socket。

下述代码片段是一个典型的创建客户端 Socket 的过程。

```
try {
    // 127.0.0.1 是 TCP/IP 协议中默认的本机地址
    Socket socket = new Socket("127.0.0.1", 4700);
} catch (IOException ioe) {
```

```
    System.out.println("Error:" + ioe);
} catch(UnknownHostException uhe) {
    System.out.println("Error:" + uhe);
}
```

2.3.2　ServerSocket 类

服务器套接字（ServerSocket）运行在服务器上，并监听在特定端口的 TCP 连接。当远程客户端的 Socket 试图与服务器指定端口建立连接时，服务器被激活，判定客户程序的连接，并打开两个主机之间固有的连接。一旦客户端与服务器建立了连接，则两者之间就可以相互传送数据。

ServerSocket 类中包含了创建 ServerSocket 对象的构造方法、在指定端口监听的方法、建立连接后发送和接收数据的方法，其常用方法及功能见表 2-6。

表 2-6　ServerSocket 类的方法列表

方法名	功能说明
public ServerSocket(int port)	构造方法，根据指定端口创建 ServerSocket 实例
public Socket accept()	这是一个阻塞方法，它停止执行代码流，并等待下一个客户端的连接。当客户端请求连接时，accept()方法返回一个 Socket 对象
public void close()	关闭当前 ServerSocket 实例
public InetAddress getInetAddress()	返回当前 ServerSocket 实例的地址信息
public int getLocalPort()	返回当前 ServerSocket 实例的服务端口

注意　在创建 ServerSocket 时如果发生错误，将产生 IOException 异常，在程序中必须对之作出处理。另外，在选择端口号时，最好选择一个大于 1023 的数以防止发生冲突。

一般情况下，ServerSocket 的工作步骤如下：

01 根据指定端口创建一个新的 ServerSocket 对象。

02 调用 ServerSocket 的 accept()方法，在指定的端口监听到来的连接。accept()一直处于阻塞状态，直到有客户端试图建立连接。这时 accept()方法返回连接客户端与服务器的 Socket 对象。

03 调用 getInputStream()方法或 getOutputStream()方法建立与客户端交互的输入/输出流。

04 服务器与客户端根据一定的协议交互，直到关闭连接。

05 关闭服务器端的 Socket。

06 回到第 2 步，继续监听下一次的连接。

下述代码片段是一个典型的创建服务器端 ServerSocket 的过程。

```
ServerSocket server = null;
try {
```

```
    // 创建一个 ServerSocket 在端口 4700 监听客户端请求
    server = new ServerSocket(4700);
} catch (IOException e) {
    System.out.println("can not listen to :" + e);
}
Socket socket = null;
try {
//accept()是一个阻塞方法，一旦有客户端请求，它就会返回一个 Socket 对象用于同客户端进行交互
    socket = server.accept();
} catch (IOException e) {
    System.out.println("Error:" + e);
}
```

2.3.3 C/S 实例

前面提到 Socket 通常用来实现 C/S 结构。使用 Socket 进行 Client/Server 程序设计的一般过程如下：

01 服务器端 Listen（监听）某个端口是否有连接请求。
02 客户端向服务器端发出 Connect（连接）请求。
03 服务器端向客户端发回 Accept（接收）消息并建立连接。
04 通过 getInputStream()和 getOutStream()方法来得到对应的输入/输出流，服务器端和客户端都可以相互读写数据。
05 关闭服务器端和客户端的 Socket。

Socket 交互编程模型如图 2-1 所示。

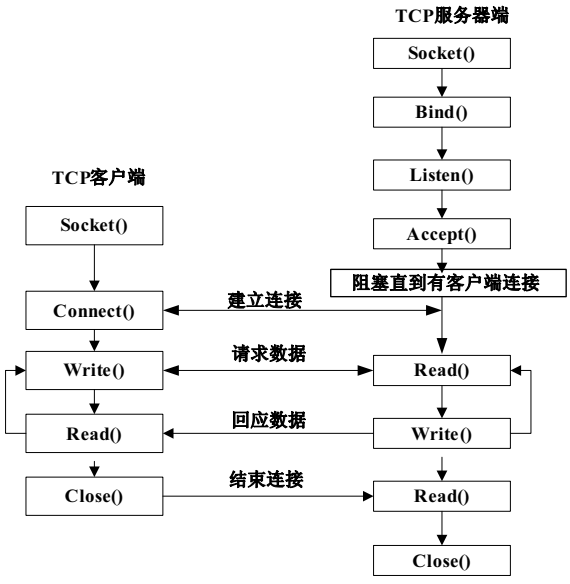


图 2–1 Socket 交互编程模型

下述代码用于实现任务描述 2.D.5，基于 Socket 通信，实现单客户端和服务器交互的 C/S 结构。

首先创建客户端程序。

【描述 2.D.5】 Client.java

```
public class Client {
    public static void main(String args[]) {
        try {
            // 向本机的 4700 端口发出客户请求
            Socket socket = new Socket("127.0.0.1", 4700);
            // 由系统标准输入构造 BufferedReader 对象
            BufferedReader sin =
            new BufferedReader(new InputStreamReader(System.in));
            // 得到输出流，并构造 PrintWriter 对象
            PrintWriter os = new PrintWriter(socket.getOutputStream());
            String readline;
            // 从系统标准输入读入一字符串
            readline = sin.readLine();
            // 若读入的字符串为 "bye"，则停止循环
            while (!readline.equals("bye")) {
                // 将读入的字符串输出到 Server
                os.println(readline);
                // 刷新输出流，使 Server 马上收到该字符串
                os.flush();
                readline = sin.readLine();
            }
            // 关闭 Socket 输出流
            os.close();
            // 关闭 Socket
            socket.close();
        } catch (Exception e) {
            System.out.println("Error" + e);
        }
    }
}
```

上述代码中，客户端将循环从系统标准输入设备中读入字符串，并输出到 Server 端，直到读入“bye”为止。

下述代码创建服务器端，用于接收客户端的数据输入。

【描述 2.D.5】 Server.java

```
public class Server {
    public static void main(String args[]) {
```

```
        try {
            ServerSocket server = null;
            try {
                // 在端口 4700 监听客户请求
                server = new ServerSocket(4700);
            } catch (Exception e) {
                System.out.println("can not listen to:" + e);
            }
            Socket socket = null;
            try {
                // 使用 accept()阻塞等待客户请求
                socket = server.accept();
            } catch (Exception e) {
                System.out.println("Error." + e);
                // 出错，打印出错信息
            }
            String line="";
            // 通过 Socket 对象的输入流构造 BufferedReader 对象
            BufferedReader is = new BufferedReader(new InputStreamReader(socket
            .getInputStream()));
            // 如果该字符串为 "bye"，则停止循环
            while (line!=null && !line.equals("bye")) {
                // 再次读入一字符串
                line = is.readLine();
                // 输出读入的字符串
                System.out.println("Server:" + line);
            }
            // 关闭 Socket 输入流
            is.close();
            // 关闭 Socket
            socket.close();
            // 关闭 ServerSocket
            server.close();
        } catch (Exception e) {
            System.out.println("Error:" + e);
        }
    }
}
```

上述代码中，服务器将循环从 Socket 输入流中读取字符串，并输出到 Server 端，直到读入“bye”为止。Socket 对象创建时，都将占用一定的资源，在 Socket 对象使用完毕时，需要将其关闭。在关闭 Socket 之前，应将与 Socket 相关的所有输入/输出流全部关闭，以释放所有的资源。而且要注意关闭的顺序，与 Socket 相关的所有输入/输出该首先关闭，然后再

调用 close()方法关闭 Socket。关闭的顺序是：

```
os.close();
is.close();
socket.close();
```

读者可以在单机上测试该程序。

以上的程序是 C/S 结构的典型工作模式，只不过在这里 Server 只能接收一个请求，接收完后 Server 就退出了。实际的应用中总是让它不停地循环接收，一旦有客户请求，Server 总是会创建一个服务线程来服务新来的客户，而自己继续监听。

2.3.4　多线程 Socket 通信

在实际应用中，服务器可以接收来自其他多个客户端的请求，提供相应的服务。为了实现在服务器方给多个客户提供服务的功能，需要对上面的程序进行改造，利用多线程实现多客户机制。服务器总是在指定的端口上监听是否有客户请求，一旦监听到客户请求，服务器就会启动一个专门的服务线程来响应该客户的请求，而服务器本身在启动完线程之后马上又进入监听状态，等待下一个客户的到来。

下述代码用于实现任务描述 2.D.6，基于线程，实现多客户端和服务器的 Socket 通信。

在多客户端和服务器通信的模型中，客户端代码无需任何调整；对于 Server 部分，需要定义线程处理 Socket 通信。

【描述 2.D.6】 MultiServer.java

```
public class MultiServer {
    public static void main(String args[]) {
        ServerSocket server = null;
        int count = 0;
        boolean listening = true;
        try {
            try {
                server = new ServerSocket(4700);
                System.out.println("Server starting....");
            } catch (Exception e) {
                System.out.println("can not listen to:" + e);
            }
            // 永远循环
            while(listening){
                // 等待客户请求
                Socket socket = server.accept();
                count++;
                System.out.println("Accept "+count+" Client!");
```

```
                // 启动客户端处理 Socket 线程
                Thread read = new ReadThread(socket,"[client"+count+"]");
                read.start();
            }
            server.close();
        } catch (Exception e) {
            System.out.println("Error:" + e);
        }
    }
}
```

上述代码中，Server 在循环中等待客户请求，一旦接收到 Socket 请求，就将该 Socket 连接的处理传入 ReadThread 中处理，其代码如下。

【描述 2.D.6】 ReadThread.java

```
public class ReadThread extends Thread {
    Socket socket = null;
    String client;
    public ReadThread(Socket socket, String client) {
        this.socket = socket;
        this.client = client;
    }
    public ReadThread(Socket socket, String side, int clientnum) {
        this(socket, side);
    }
    public void run() {
        try {
            String line="";
            BufferedReader is = new BufferedReader(new InputStreamReader(socket
            .getInputStream()));
            // 如果该字符串为 "bye"，则停止循环
            while (line!=null && !line.equals("bye")) {
                line = is.readLine();
                System.out.println(client+" read :" + line);
            }
            // 关闭 Socket 输入流
            is.close();
            // 关闭 Socket
            socket.close();
            // 关闭 ServerSocket
        } catch (Exception e) {
            System.out.println("Error : " + e);
        }
```

```
    }
}
```

ReadThread 继承自 Thread，有一个 Socket 成员变量 socket，用于接收 Server 传递的与客户端对应的本地 Socket 对象，并在线程体内读取客户端写入的数据。

小结

通过本章的学习，学生应该能够学会：

- IP 协议是一个无连接、不可靠的协议。
- 域名是由一串用点分隔的名字组成的 Internet 上某一台计算机或计算机组的名称。
- DNS（Domain Name Server）是进行域名解析的服务器。
- URL（Uniform Resource Locator）是统一资源定位器的简称。
- URL 的组成：协议名://机器名＋端口号＋文件名＋内部引用。
- URLConnection 是一个抽象类，代表与 URL 指定的数据源的动态连接。
- 网络上的两个程序通过 Socket 实现双向通信和数据交换。
- Java 提供了两个类 Socket 和 ServerSocket，分别用来表示双向连接的客户端和服务端。
- 在创建 Socket 或 ServerSocket 时必须捕获或抛出异常。
- 在 Socket 对象使用完毕时，要将其关闭，并且遵循一定的关闭次序。

练习

1. 下面合法的 IP 地址是_____。
 A. 192.168.0.1　　B. 192.168.0.256
 C. 192.168.-1.255　　D. 202.102.56.27.1
2. HTTP 服务的端口是_____。
 A. 21　　B. 23　　C. 25　　D. 80
3. 套接字包括_____。
 A. 端口号　　B. IP 地址
 C. 端口号和 IP 地址　　D. 都不是
4. 等待客户机请求连接，服务器可以使用的类是_____。
 A. Socket　　B. ServerSocket
 C. Server　　D. URL
5. ServerSocket 的 accept()方法返回的对象类型是_____。
 A. Socket　　B. ServerSocket
 C. Server　　D. URL

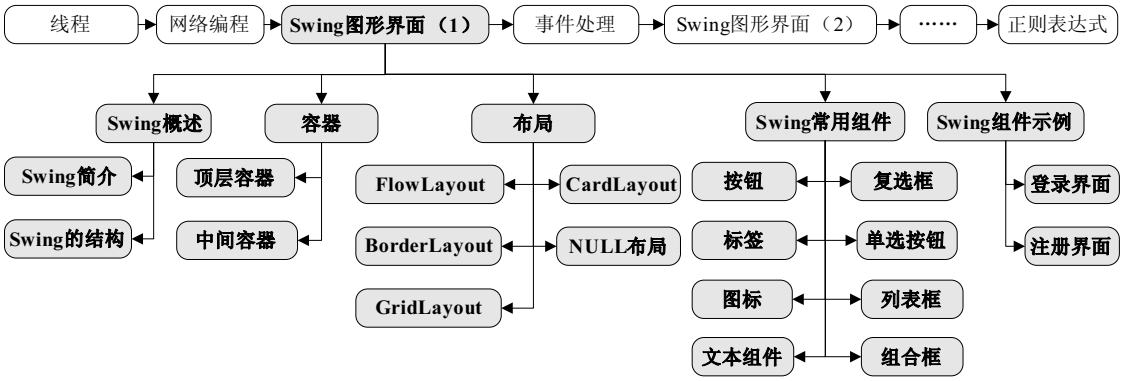
6. TCP 用于_____。
 A. 根据点分八位组或域名来识别 Internet 上的机器
 B. 确保数据包按照发送的顺序到达
 C. A 和 B
 D. 都不是
7. 用来封装计算机 IP 地址和域名的类是_____。
8. 使用 Socket 进行客户端和服务器通信，具体要求如下：
 ① 客户端向服务器发送“我是***”，其中***是用户姓名。例如：“我是张三”。
 ② 服务器接收用户名并在控制台中显示；服务器统计接收信息的个数，向客户端返回“您是第*个访问服务器的人!”（*是统计数）。

第 3 章　Swing 图形界面（1）

本章目标

- 理解 Java Swing 的基本结构
- 掌握 Java 容器的使用方式
- 掌握 Java 常用布局的使用方式
- 掌握 Swing 基础组件的使用方式
- 使用 Swing 组件创建并布局登录界面
- 使用 Swing 组件创建并布局注册界面

学习导航



任务描述

【描述 3.D.1】

通过实例演示顶层容器（JFrame）和中间容器（JPanel）的使用。

【描述 3.D.2】

通过实例演示常用布局管理器 FlowLayout、BorderLayout、CardLayout 和 NULL 布局的使用。

【描述 3.D.3】

使用网格布局 GridLayout 来实现计算器的界面布局。

【描述 3.D.4】

通过实例演示常用 Icon 组件的使用。

【描述 3.D.5】

创建一个用户登录界面，登录信息有用户名和密码。

【描述 3.D.6】

创建一个注册界面，注册信息有用户名、密码、确认密码、性别、爱好、地址和学历。

3.1　Swing 概述

到目前为止，所编写的程序都是通过键盘接收输入，而在控制台屏幕上显示结果的，现代的程序早已不采用这种操作模式了，网络程序更是如此。从本章开始，将介绍如何编写具有图形用户界面（GUI）的 Java 程序。

3.1.1　Swing 简介

在 Java 1.0 发布时，包含了一个用于基本 GUI 程序设计的类库，Sun 将它们称为抽象窗口工具箱（Abstract Windows Toolkit，AWT）。AWT 为 Java 应用程序提供了基本的图形组件，使用 AWT 构建的 GUI 程序运行在不同的平台上时会有不同的外观效果，因此，要想给予用户一致的、可预见性的界面操作方式，是相当困难的。

1996 年，Netscape 创建了一种称为 IFC（Internet Foundation Class）的 GUI 库，它采用了与 AWT 完全不同的工作方式。Netscape 的 IFC 组件在程序运行的所有平台上的外观和动作都一样。Sun 与 Netscape 合作完善了这种方式，创建名为 Swing 的 GUI 库。Swing 组件完全是用 Java 编写的，不需要使用那些当前平台所用的复杂的 GUI 功能，因此，使用 Swing 构建的 GUI 应用程序运行在不同的平台上时显示一样的外观效果。

Swing 没有完全代替 AWT，而是基于 AWT 架构之上。Swing 仅仅提供了能力更加强大的用户界面组件。尤其在采用 Swing 编写的程序中，还需要使用基本的 AWT 处理事件。

Swing 的优点有以下几个方面：

- Swing 拥有一个丰富、便捷的用户界面元素集合。
- Swing 对底层平台依赖很少，因此与平台相关的 bug 很少。
- Swing 给予不同平台的用户一致的感观效果。

3.1.2　Swing 的结构

Swing 组件位于 javax.swing 包中，javax 是一个 Java 扩展包。要有效地使用 GUI 组件，必须理解 javax.swing 和 java.awt 包中的继承层次，尤其是要理解 Component 类、Container 类和 JComponent 类，这三个类声明了大多数 Swing 组件的通用特性。图 3-1 显示了 Swing 组件的继承层次：Object 类是 Java 类层次的父类；Component 类是 Object 类的子类；Container 类是 Component 类的子类；JComponent 类是 Container 的子类。

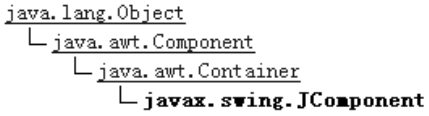


图 3-1　Swing 组件的继承层次

javax.swing 包中提供的 Swing 组件有很多，如图 3-2 所示列举出部分 Swing 组件及相互间的继承关系：



图 3–2　Swing 组件继承图

3.2 容器

容器是用来放置图形组件的一个对象，从图 3-1 中可以看到容器（Container）实际上是 Component 类的子类，因此容器类对象本身也是一个组件，具有组件的性质，同时还具备容纳其他组件和容器的功能。

3.2.1　顶层容器

顶层容器就是不包含在其他容器中的容器，Swing 中常用的顶层容器有 JFrame。JFrame 被称为窗口框架，它扩展了 java.awt.Frame 类，其常用的方法及功能见表 3-1。

表 3–1　JFrame 的方法列表

方法	功能说明
JFrame()	构造方法，用于创建一个初始时不可见的新窗体
JFrame(String title)	构造方法，用于创建一个新的、初始不可见的、具有指定标题的窗体
frameInit()	在构造方法中调用该方法来初始化窗体
add(Component comp)	添加组件
setLocation(int x,int y)	设置位置坐标，以像素为单位
setSize(int width,int height)	设置大小，以像素为单位
setVisible(boolean b)	设置是否可视，当参数是 true 则可视，当是 false 则隐藏
setContentPane(Container contentPane)	设置容器面板
setIconImage(Image image)	设置窗体左上角的图标
setDefaultCloseOperation(int operation)	设置窗体默认的关闭操作，参数是常量，如 EXIT_ON_CLOSE
setTitle(String title)	设置标题

下述代码用于实现任务描述 3.D.1，通过 JFrame 创建可视化窗体。

【描述 3.D.1】 FrameDemo.java

```
public class FrameDemo extends JFrame {
    public FrameDemo() {
        super("我的窗口");
        this.setSize(300, 200);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]) {
        FrameDemo frame = new FrameDemo();
        frame.setVisible(true);
    }
}
```

在上述代码中，MyFrame 继承了 JFrame，在该类的构造方法中使用如下语句

```
super("我的窗口");
```

上述语句用来构造一个标题为“我的窗口”的窗体。

使用如下语句

```
this.setSize(300, 200);
```

来设置窗体的宽度为 300px，高度为 200px（px 为像素单位）。

使用如下语句

```
setLocation(100,100);
```

来设置窗体在显示器屏幕中的 *x*、*y* 轴坐标都为 100px。然后在 main()方法中先实例化一个 MyFrame 窗体对象，再通过 setVisible(true)方法让窗体对象显示。

运行结果如图 3-3 所示：



图 3–3　MyFrame 的运行结果

3.2.2　中间容器

中间容器不能作为顶层容器，它必须包含在其他容器中。例如，可以将中间容器放到 JFrame 顶层容器中。JPanel 是一个中间容器，通常称为“面板”。JPanel 类继承自 JComponent 类，其常用的方法及功能见表 3-2。

表 3–2　JPanel 的方法列表

方法	功能说明
JPanel()	构造方法，用于创建一个默认布局为流布局的面板
JPanel((LayoutManager layout)	构造方法，用于创建一个指定布局的面板
add(Component comp)	添加组件
setLayout(LayoutManager mgr)	设置面板的布局

下述代码用于实现任务描述 3.D.1，通过中间容器 JPanel 创建带两个按钮的可视化窗体。

【描述 3.D.1】 PanelDemo.java

```
public class PanelDemo extends JFrame {
    // 声明面板对象
    private JPanel jp;
    // 声明按钮对象
    private JButton b1;
    private JButton b2;
    public PanelDemo() {
        // 创建一个标题为"测试面板"的窗口
        super("测试面板");
        // 实例化面板对象
```

```
        jp = new JPanel();
        // 实例化一个按钮对象，该按钮上的文本为"Button 1"
        b1 = new JButton("Button 1");
        // 实例化一个按钮对象，该按钮上的文本为"Button 2"
        b2 = new JButton("Button 2");
        // 将按钮添加到面板中
        jp.add(b1);
        jp.add(b2);
        // 将面板添加到窗体框架中
        this.add(jp);
        this.setSize(300, 100);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        PanelDemo sp = new PanelDemo();
        sp.setVisible(true);
    }
}
```

在上述代码中，先在类体内声明用到的各个组件，如面板对象 jp、按钮对象 b1 和 b2。然后在该类的构造方法中实例化各个组件（JButton 是按钮类，该类的详细介绍见后续章节）。

运行结果如图 3-4 所示。



图 3–4　运行结果

3.3　布局

布局管理器用来管理组件在容器中的布局格式。当容器中容纳多个组件时，可以使用布局管理器将这些组件安排在一个容器中。布局管理器在 java.awt 包中，该包中常用的布局管理器有：FlowLayout、BorderLayout、GridLayout 和 CardLayout 等。

3.3.1　FlowLayout

FlowLayout 称为“流布局”，将组件从左到右顺序、流动地安排到容器中，直到占满上方的空间时，则向下移动一行，继续流动。FlowLayout 是面板的默认布局，其常用的构造方法及功能见表 3-3。

表 3–3　FlowLayout 的构造方法列表

方法	功能说明
FlowLayout()	创建一个中间对齐、默认间距为 5 像素的新流布局管理器
FlowLayout(int align)	创建一个指定对齐方式、默认间距为 5 像素的新流布局管理器
FlowLayout(int align,int hgap,int vgap)	创建一个指定对齐方式、水平和垂直间距的新流布局管理器

下述代码用于实现任务描述 3.D.2，演示流布局 FlowLayout 的使用。

【描述 3.D.2】 FlowLayoutDemo.java

```
public class FlowLayoutDemo extends JFrame {
    private JPanel p;
    private JButton b1, b2, b3;
    public FlowLayoutDemo() {
        super("流布局");
        p = new JPanel();
        //创建一个流布局对象，对齐方式是左对齐，水平间距 10 像素，垂直间距 15 像素
        FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 15);
        //设置面板的布局
        p.setLayout(layout);
        b1 = new JButton("Button 1");
        b2 = new JButton("Button 2");
        b3 = new JButton("Button 3");
        p.add(b1);
        p.add(b2);
        p.add(b3);
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        PanelDemo sp = new PanelDemo();
        sp.setVisible(true);
    }
}
```

在上述代码中，创建好流布局对象后，使用 setLayout()方法将该布局对象设置到面板中。运行结果如图 3-5 所示。



图 3–5　FlowLayout 运行结果

注意　采用流布局，当改变窗体的大小时，可以发现各组件的位置随着窗体的变化而变化，而且各组件大小不变，保持原来的“最合适”大小。

3.3.2　BorderLayout

BorderLayout 称为“边界布局”，它允许将组件有选择地放置到容器的中部、北部、南部、东部或西部，如图 3-6 所示。

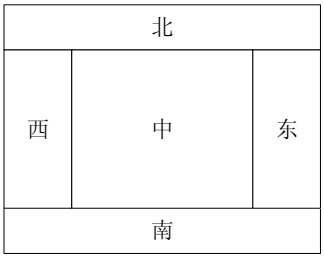


图 3–6　边界布局

BorderLayout 是窗体框架（JFrame）的默认布局，其常用的构造方法及功能见表 3-4。

表 3–4　BorderLayout 的构造方法列表

方法	功能说明
BorderLayout()	创建一个新的、无间距的边界布局管理器对象
BorderLayout(int hgap,int vgap)	创建一个新的、指定水平、垂直间距的边界布局管理器

下述代码用于实现任务描述 3.D.2，演示边界布局 BorderLayout 的使用。

【描述 3.D.2】 BorderLayoutDemo.java

```
public class BorderLayoutDemo extends JFrame {
    private JPanel p;
    private JButton b1, b2, b3, b4, b5;
    public BorderLayoutDemo() {
        super("边界布局");
        p = new JPanel();
        // 创建一个边界布局管理器对象，并将该布局设置到面板中
        p.setLayout(new BorderLayout());
        b1 = new JButton("Button 1");
        b2 = new JButton("Button 2");
        b3 = new JButton("Button 3");
        b4 = new JButton("Button 4");
        b5 = new JButton("Button 5");
        //将按钮放置到面板中的指定位置
        p.add(b1, BorderLayout.EAST);
        p.add(b2, BorderLayout.WEST);
```

```
        p.add(b3, BorderLayout.SOUTH);
        p.add(b4, BorderLayout.NORTH);
        p.add(b5, BorderLayout.CENTER);
        this.add(p);
        this.setSize(300, 200);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        BorderLayoutDemo f = new BorderLayoutDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 BorderLayout 创建一个边界布局管理器对象，同时将其设置到面板中。因为边界布局管理器有五个位置，所以将组件添加到容器中时需要指明该组件所放置的位置。在 BorderLayout 类中定义了五个表明位置的常量，分别为：EAST（东）、WEST（西）、SOUTH（南）、NORTH（北）和 CENTER（中）。如果没有指明位置，则放置在中部（默认位置）。

运行结果如图 3-7 所示。

注意 采用边界布局，当改变窗体的大小时，可以发现东西南北四个位置上的组件长度进行拉伸，而中间位置的组件进行扩展。



图 3–7　BorderLayout 运行结果

3.3.3　GridLayout

GridLayout 称为“网格布局”，它像表格一样，按行和列排列所有组件，且每个单元格大小都一样；在向表格里面添加组件的时候，它们将按照从左到右、从上到下的顺序加入。

网格布局常用的构造方法及功能见表 3-5。

表 3-5　GridLayout 的构造方法列表

方法	功能说明
GridLayout(int rows,int cols)	创建一个新的、指定行数和列数的网格布局管理器
GridLayout(int rows,int cols,int hgap,int vgap)	创建一个新的、指定行数、列数、水平间距和垂直间距的网格布局管理器

下述代码用于实现任务描述 3.D.3，使用网格布局来实现计算器的界面布局。

【描述 3.D.3】 Calculator.java

```
public class Calculator extends JFrame {
    // 声明一个文本栏控件，用于显示计算结果
    private JTextField txtResult;
    private JPanel p;
    // 定义一个字符串数组，将计算器中按钮的文字都放在该数组中
    private String name[] = { "7", "8", "9", "+", "4", "5", "6", "-", "1", "2",
            "3", "*", "0", ".", "=", "/" };
    // 声明一个按钮数组，该数组的长度以字符串数组的长度为准
    private JButton button[] = new JButton[name.length];
    public Calculator() {
        super("计算器");
        // 实例化文本栏控件
        txtResult = new JTextField(20);
        // 将文本栏控件放置在窗体框架的上方（北部）
        this.add(txtResult, BorderLayout.NORTH);
        // 实例化面板对象，同时设置此面板布局为 4 行 4 列的网格布局
        p = new JPanel(new GridLayout(4, 4));
        // 循环实例化按钮对象数组
        for (int i = 0; i < button.length; i++) {
            button[i] = new JButton(name[i]);
            p.add(button[i]);
        }
        this.add(p, BorderLayout.CENTER);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        Calculator f = new Calculator();
        f.setVisible(true);
    }
}
```

上述代码中，使用边界布局和网格布局共同构造了计算器界面，因为边界布局是窗体框

架的默认布局，因此不必创建边界布局管理器对象。在实例化面板对象时，可将布局管理器作为参数，如下所示实例化面板对象的同时，设置此面板布局为 4 行 4 列的网格布局：

```
p = new JPanel(new GridLayout(4, 4));
```

其中，JTextField 是文本框类，有关该类的详细介绍见后续章节。

运行结果如图 3-8 所示。



图 3–8　Calculator 运行结果

注意　采用网格布局，当改变窗体的大小时，可以发现各组件的大小随着窗体的变化而均匀变化。

3.3.4　CardLayout

CardLayout 称为“卡片布局”，它将加入到容器的组件看成一叠卡片，只能看到最上面的组件。通过调用 CardLayout 的一些方法，才能显示其中的组件。其常用的方法及功能见表 3-6。

表 3–6　CardLayout 的方法列表

方法	功能说明
CardLayout()	构造方法，用于创建一个默认间距为 0 的新卡片布局管理器
CardLayout(int hgap,int vgap)	构造方法，用于创建一个指定水平和垂直间距的新卡片布局管理器
first(Container parent)	显示第一张卡片
last(Container parent)	显示最后一张卡片
previous(Container parent)	显示当前卡片的上一张卡片
next(Container parent)	显示当前卡片的下一张卡片
show(Container parent,String name)	显示指定名称的卡片

下述代码用于实现任务描述 3.D.2，演示卡片布局 CardLayout 的使用。

【描述 3.D.2】 CardLayoutDemo.java

```
public class CardLayoutDemo extends JFrame {
    private JPanel p;
```

```
    private JButton b1, b2, b3, b4;
    // 声明卡片布局管理器
    private CardLayout cl;
    public CardLayoutDemo() {
        super("卡片布局");
        // 实例化卡片布局管理器对象
        cl = new CardLayout();
        // 实例化面板对象，其布局为卡片布局
        p = new JPanel(cl);
        b1 = new JButton("Button 1");
        b2 = new JButton("Button 2");
        b3 = new JButton("Button 3");
        b4 = new JButton("Button 4");
        // 将组件添加到面板中， 此时 add()带两个参数，分别是名称和组件
        p.add("1", b1);
        p.add("2", b2);
        p.add("3", b3);
        p.add("4", b4);
        this.add(p);
        // 显示最后一张卡片，此代码中最后一张卡片盛放的是 b4,结果显示"Button 4"
        // cl.last(p);
        // 显示名称是 2 的卡片，结果显示"Button 2"
        // cl.show(p, "2");
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        CardLayoutDemo f = new CardLayoutDemo();
        f.setVisible(true);
    }
}
```

上述代码中使用 CardLayout 布局，需要注意的是：如果容器的布局是卡片布局，往容器中用 add()方法添加组件时带两个参数，分别指明名称标识和组件对象。

运行结果如图 3-9 所示。



图 3–9　CardLayout 运行结果

分别将代码中已注释的行

```
// cl.last(p);
```

和

```
// cl.show(p, "2");
```

取消注释，观察运行结果。

注意 采用卡片布局，当改变窗体的大小时，可以发现卡片上的组件大小随着窗体的变化而变化，并始终占满整个窗口。

3.3.5 NULL 布局

在实际开发过程中，用户界面比较复杂，而且要求美观，使用 FlowLayout、BorderLayout、GridLayout 及 CardLayout 这些布局很难满足要求。这时可以采用 NULL（空）布局，即容器不采用任何布局，而是通过设置每个组件在容器中的位置及大小来安排。

使用 NULL 布局的步骤如下：

01 设置容器的布局为空，例如：

```
p.setLayout(null);//设置面板对象的布局为空
```

02 设置组件的位置及大小。

```
b.setBounder(30,60,40,25);//设置按钮对象的 x 轴坐标为 30，y 轴坐标为 60，宽度 40，高度 25
```

03 将组件添加到容器中。

```
p.add(b);//将按钮添加到面板中
```

下述代码用于实现任务描述 3.D.2，演示空布局 NULL 的使用。

【描述 3.D.2】 NullDemo.java

```
public class NullDemo extends JFrame {
    private JPanel p;
    private JButton b1, b2;
    public NullDemo() {
        super("空布局");
        p = new JPanel();
        // 设置面板布局为空
        p.setLayout(null);
        b1 = new JButton("确定");
        b2 = new JButton("取消");
```

```
        // 设置按钮位置大小
        b1.setBounds(30, 60, 60, 25);
        b2.setBounds(100, 60, 60, 25);
        // 将按钮添加到面板中
        p.add(b1);
        p.add(b2);
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        NullDemo f = new NullDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，面板使用空布局，两个按钮在面板中的位置和大小使用绝对位置进行固定。

运行结果如图 3-10 所示。



图 3-10　NullDemo 的运行结果

注意　NULL 布局一般用于组件之间位置相对固定，并且窗口不允许随便变换大小的情况，否则当窗口大小发生变化时，因所有组件使用绝对位置，会产生组件整体“偏移”的情况。

3.4　Swing 常用组件

任何一个复杂的 GUI 都是由最基本的组件在布局的统一控制下组合而成的，Java 中常用的组件有：按钮、标签、图标、文本组件、复选框、单选按钮、列表框和组合框等。

3.4.1　按钮

JButton 类提供一个能够被点击的按钮功能，当单击按钮时，按钮将处于“下压”形状，松开后又恢复原状。在按钮中可以显示图标、字符串或两者同时显示。JButton 类的常用方法

及功能见表 3-7。

表 3–7　JButton 类的方法列表

方法	功能说明
JButton(String str)	构造方法，用于创建一个指定文本的按钮组件
JButton(Icon i)	构造方法，用于创建一个指定图标的按钮组件
JButton(String str,Icon icon)	构造方法，用于创建一个指定文本和图标的按钮组件
String getText()	获取按钮上的文本内容
void setText(String str)	设置按钮上的文本内容
void setIcon(Icon icon)	设置按钮上的图标

3.4.2　标签

JLabel 类可以显示文字或图标，它没有边界，也不响应用户输入。可以利用标签标识组件，起到提示的作用。例如，使用标签标识文本框、文本域等这种不带标签的组件。

JLabel 类的常用的方法及功能见表 3-8。

表 3–8　JLabel 类的方法列表

方法	功能说明
JLabel(Icon icon)	构造方法，用于创建一个指定图标的标签对象
JLabel(String text)	构造方法，用于创建一个指定文本的标签对象
JLabel(String text, Icon icon, int horizontalAlignment)	构造方法，用于创建一个指定文本、图标和对齐方式的标签对象
void setText(String txt)	设置标签中的文本内容
void setIcon(Icon icon)	设置标签中的图标
String getText()	获取标签中的文本内容

3.4.3　图标

Icon 图标用于加载图片，加载的图片文件格式一般限于 GIF 和 JPG 文件。Icon 是一个接口，ImageIcon 类实现该接口，创建一个 ImageIcon 类的对象时只需将图片名作为参数，就会打开文件并得到图形。

下述代码用于实现任务描述 3.D.4，演示 Icon 组件的使用。

【描述 3.D.4】 IconDemo.java

```
public class IconDemo extends JFrame {
    private JPanel p;
    private JButton b;
    private JLabel lbl;
    public IconDemo() {
        super("图片");
```

```
        p = new JPanel();
        Icon icon1=new ImageIcon("d:\\javalogo.gif");
        lbl=new JLabel(icon1);
        Icon icon2=new ImageIcon("d:\\save.gif");
        b=new JButton("保存",icon2);
        p.add(lbl);
        p.add(b);
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        IconDemo f = new IconDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 ImageIcon 实例化图片对象时，图片文件名包含路径，“\\”是转义字符，代表“\”。运行结果如图 3-11 所示。



图 3–11　IconDemo 运行结果

3.4.4　文本组件

在 Java 中，文本组件主要用于实现文本串的输入，常用的文本组件有：

- 文本框（JTextField）
- 文本域（JTextArea）
- 密码框（JPasswordField）

其中，文本框只能接收单行的文本输入，而文本域可以接收多行的文本输入，这两个类都继承于 JTextComponent 类。JTextComponent 是抽象类，此类定义了文本组件的通用方法及特性，不能直接构造该抽象类的对象。

JTextField 是实现文本框组件的类，该类与 java.awt.TextField 具有源代码兼容性，并具有建立字符串的方法，此字符串用做针对被激发的操作事件的命令字符串。JTextField 类的常用方法及功能见表 3-9。

表 3-9　JTextField 类的方法列表

方法	功能说明
JTextField(int cols)	构造方法，用于创建一个空文本的、指定长度的文本框
JTextField(String str)	构造方法，用于创建一个指定文本的文本框
JTextField(String s, int cols)	构造方法，用于创建一个指定文本的、指定长度的文本框
String getText()	获取文本框中的文本内容
void setText(String str)	设置文本框中的文本内容

JTextArea 是实现文本域组件的类，是一个显示纯文本的多行区域。该组件中可以编辑多行多列文本，具有换行能力，其常用的方法及功能见表 3-10。

表 3-10　JTextArea 类的方法列表

方法	功能说明
JTextArea(int rows, int columns)	构造方法，用于创建一个空文本、指定行数和列数的文本域
JTextArea(String text)	构造方法，用于创建一个指定文本的文本域
JTextArea(String text,int rows,int columns)	构造方法，用于添加组件创建一个指定文本、指定行数和列数的文本域
String getText()	获取文本域中的文本内容
void setText(String str)	设置文本域中的文本内容

JPasswordField 是实现密码框组件的类，可以提供让用户输入密码的功能。JPasswordField 继承于 JTextField，允许编辑单行文本，其视图指示输入内容，但不显示原始字符。即用户在密码框中输入的密码都以特殊符号显示。JPasswordField 类的常用方法及功能见表 3-11。

表 3-11　JPasswordField 类的方法列表

方法	功能说明
JPasswordField (int cols)	构造方法，用于创建一个空的、指定长度的密码框
JPasswordField (String str)	构造方法，用于创建一个指定密码信息的密码框
JPasswordField (String s, int cols)	构造方法，用于创建一个指定密码信息的、指定长度的密码框
char[] getPassword()	获取密码框中的密码，以字符型数组形式返回
setEchoChar(char c)	设置密码框中的反射字符为指定字符，JDK1.6 中密码框的默认反射字符为‘.’

3.4.5　复选框

复选框（JCheckBox）可以控制选项的开启或关闭，在复选框上单击时，可以改变复选框的状态，复选框可以被单独使用或作为一组使用。其常用的方法及功能见表 3-12。

表 3-12　JCheckBox 的方法列表

方法	功能说明
JCheckBox(String str)	构造方法，用于创建一个指定文本的、没有被选中的复选框

（续表）

方法	功能说明
JCheckBox(String str, boolean state)	构造方法，用于创建一个指定文本和是否选中的复选框。其中 state 如果为 true，复选框在初始化时状态为被选中，否则相反
void setSelected(boolean state)	设置复选框的选中状态。true 为选中，false 则不被选中
boolean isSelected()	获得复选框是否被选中。返回 true 为选中，返回 false 则不被选中

3.4.6　单选按钮

单选按钮（JRadioButton）可被选择或取消选择，并可为用户显示其状态。单选按钮一般成组出现，与按钮组（ButtonGroup）对象配合使用，一次只能选择其中的一个按钮。

注意　使用单选按钮时，要创建一个 ButtonGroup 对象并用其 add()方法将 JRadioButton 对象包含到此组中。ButtonGroup 对象为逻辑分组，而不是物理分组，因此仍要将 JRadioButton 对象添加到容器对象中。

JRadioButton 类中的常用方法及功能见表 3-13。

表 3-13　JRadioButton 类的方法列表

方法	功能说明
JRadioButton(String str)	构造方法，用于创建一个指定文本的、没有被选中的单选按钮
JRadioButton(String str,boolean state)	构造方法，用于创建一个指定文本的、是否被选中的单选按钮
void setSelected(boolean state)	设置单选按钮的选中状态：true 为选中，false 则不被选中
boolean isSelected()	获得单选按钮是否被选中：返回 true 为选中，false 则不被选中

下述代码用于实现任务描述 3.D.4，演示单选按钮的使用。

【描述 3.D.4】RadioButtonDemo.java

```
public class RadioButtonDemo extends JFrame {
    private JPanel p;
    private JRadioButton rb1, rb2;
    private ButtonGroup bg;
    public RadioButtonDemo() {
        super("单选");
        p = new JPanel();
        // 创建单选按钮
        rb1 = new JRadioButton("男", true);
        rb2 = new JRadioButton("女");
        // 创建按钮组
        bg = new ButtonGroup();
        // 将 rb1 和 rb2 两个单选按钮添加到按钮组中,这两个单选按钮只能选中其一
```

```
        bg.add(rb1);
        bg.add(rb2);
        p.add(rb1);
        p.add(rb2);
        this.add(p);
        this.setSize(200, 100);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        RadioButtonDemo f = new RadioButtonDemo();
        f.setVisible(true);
    }
}
```

上述代码使用 JRadioButton 创建单选按钮，将单选按钮添加到 ButtonGroup 中实现单选规则。

运行结果如图 3-12 所示。



图 3-12　RadioButtonDemo 运行结果

3.4.7　列表框

JList 类提供列表框功能。在列表框中可以提供多个选择项，允许用户选择一个或多个（按住“Ctrl”键才能选中多个）。JList 类的常用方法及功能见表 3-14。

表 3-14　JList 类的方法及功能

方法	功能说明
JList()	构造方法，用于创建一个没有选项的列表框
JList(Object[] listData)	构造方法，用于创建一个列表框，其选项列表为对象数组中的元素
JList(Vector<?> listData)	构造方法，用于创建一个列表框，其选项列表为泛型集合中的元素
int getSelectedIndex()	获得选中选项的下标，此时用户选择一个选项
Object getSelectedValue()	获得列表中用户选中的选项的值
Object[] getSelectedValues()	以对象数组的形式返回所有被选中选项的值
void addItem(Object obj)	添加一个新的选项

3.4.8　组合框

JComboBox 类提供组合框功能，它是一个文本框和下拉列表的组合。用户可以在下拉列表选项中选择一个不同的选项或在文本框内输入选择项。其常用的方法及功能见表 3-15。

表 3–15　JComboBox 常用的方法及功能

方法	功能说明
JComboBox ()	构造方法，用于创建一个没有选项的组合框
JComboBox (Object[] listData)	构造方法，用于创建一个组合框，其选项列表为对象数组中的元素
JComboBox (Vector<?> listData)	构造方法，用于创建一个组合框，其选项列表为泛型集合中的元素
int getSelectedIndex()	获得组合框中用户选中选项的下标
Object getSelectedValue()	获得组合框中用户选中选项的值
void addItem(Object obj)	添加一个新的选项

3.5　Swing 组件示例

3.5.1　登录界面

下述代码用于实现任务描述 3.D.5，使用 Swing 组件创建一个用户登录界面。

【描述 3.D.5】Login.java

```
public class Login extends JFrame {
    private JPanel p;
    private JLabel lblName, lblPwd;
    private JTextField txtName;
    private JPasswordField txtPwd;
    private JButton btnOk, btnCancle;
    public Login() {
        super("登录");
        p = new JPanel();
        p.setLayout(null);
        lblName = new JLabel("用户名");
        lblPwd = new JLabel("密码");
        txtName = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtPwd.setEchoChar('*');
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 120, 25);
```

```
        lblPwd.setBounds(30, 60, 60, 25);
        txtPwd.setBounds(95, 60, 120, 25);
        btnOk.setBounds(60, 90, 60, 25);
        btnCancle.setBounds(125, 90, 60, 25);
        p.add(lblName);
        p.add(txtName);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(btnOk);
        p.add(btnCancle);
        this.add(p);
        this.setSize(250, 170);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        Login f = new Login();
        f.setVisible(true);
    }
}
```

在上述代码中，面板使用空布局，需要设置各组件的坐标和大小。

运行结果如图 3-13 所示。



图 3-13　Login 运行结果

3.5.2　注册界面

下述代码用于实现任务描述 3.D.6，创建一个用户注册界面。

【描述 3.D.6】Regist.java

```
public class Regist extends JFrame {
    private JPanel p;
```

```
private JLabel lblName, lblPwd, lblRePwd, lblSex, lblLike, lblAddr,
        lblDegree;
private JTextField txtName;
private JPasswordField txtPwd, txtRePwd;
private JRadioButton rbMale, rbFemale;
private JCheckBox ckbRead, ckbNet, ckbSwim, ckbTour;
private JTextArea txtAddr;
private JComboBox cmbDegree;
private JButton btnOk, btnCancle;
public Regist() {
    super("用户注册");
    p = new JPanel(new GridLayout(8, 1));
    lblName = new JLabel("用户名:");
    lblPwd = new JLabel("密    码:");
    lblRePwd = new JLabel("确认密码:");
    lblSex = new JLabel("性别:");
    lblLike = new JLabel("爱好:");
    lblAddr = new JLabel("地址:");
    lblDegree = new JLabel("学历:");
    txtName = new JTextField(20);
    txtPwd = new JPasswordField(20);
    txtRePwd = new JPasswordField(20);
    rbMale = new JRadioButton("男");
    rbFemale = new JRadioButton("女");
    ckbRead = new JCheckBox("阅读");
    ckbNet = new JCheckBox("上网");
    ckbSwim = new JCheckBox("游泳");
    ckbTour = new JCheckBox("旅游");
    txtAddr = new JTextArea(3, 20);
    String str[] = { "小学", "中学", "高中", "本科", "研究生" };
    cmbDegree = new JComboBox(str);
    // 设置组合框可编辑
    cmbDegree.setEditable(true);
    btnOk = new JButton("确定");
    btnCancle = new JButton("取消");
    ButtonGroup bg = new ButtonGroup();
    bg.add(rbMale);
    bg.add(rbFemale);
    JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    p1.add(lblName);
    p1.add(txtName);
```

```
        p.add(p1);
        JPanel p2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p2.add(lblPwd);
        p2.add(txtPwd);
        p.add(p2);
        JPanel p3 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p3.add(lblRePwd);
        p3.add(txtRePwd);
        p.add(p3);
        JPanel p4 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p4.add(lblSex);
        p4.add(rbMale);
        p4.add(rbFemale);
        p.add(p4);
        JPanel p5 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p5.add(lblLike);
        p5.add(ckbRead);
        p5.add(ckbNet);
        p5.add(ckbSwim);
        p5.add(ckbTour);
        p.add(p5);
        JPanel p6 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p6.add(lblAddr);
        p6.add(txtAddr);
        p.add(p6);
        JPanel p7 = new JPanel(new FlowLayout(FlowLayout.LEFT));
        p7.add(lblDegree);
        p7.add(cmbDegree);
        p.add(p7);
        JPanel p8 = new JPanel(new FlowLayout(FlowLayout.CENTER));
        p8.add(btnOk);
        p8.add(btnCancle);
        p.add(p8);
        this.add(p);
        this.setSize(305, 350);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
```

```
        Regist f = new Regist();
        f.setVisible(true);
    }
}
```

在上述代码中，混合使用网格布局和流布局。面板对象 p 是网格布局，8 行 1 列。8 个小面板是流布局，分别放在不同的行中，其他的组件创建后放在相应的小面板中。

运行结果如图 3-14 所示。



图 3–14　Regist 运行结果

小结

通过本章的学习，学生应该能够学会：

- 尽管 Swing 独立于 AWT，但它是依照基本的 AWT 类实现的。
- Container 对象可用于将组件组合在一起。
- 容器中的组件根据特定布局排列。
- Swing 中的容器有顶级容器和中间容器两种。
- 中间容器可以相互嵌套以实现复杂的界面布局。
- 布局管理器类：FlowLayout、BorderLayout、GridLayout、CardLayout。
- 文本组件：JTextField、JTextArea、JPasswordField。
- 标签：JLabel。
- 按钮：JButton。
- 图标：Icon 接口、ImageIcon 类。
- 复选框：JCheckBox。
- 单选按钮：JRadioButton，使用 ButtonGroup 实现单选规则。

- 列表：JList。
- 组合框：JComboBox。

练习

1. Swing 组件位于_____包中。
 A. java.swing　　B. java.awt　　C. javax.swing　　D. java.util
2. 下面_____管理器是居中放置组件的，当同一行超出容器宽度后才会从新行开始放置组件。
 A. 流布局　　B. 网络布局　　C. 边界布局　　D. 卡片布局
3. 使用边界布局管理器时，_____区域会自动垂直调整大小，而不在水平方向上调整。
 A. 上或下　　B. 左或右　　C. 中间　　D. 任何区域
4. 利用边界布局，向容器中间添加一个组件，其中容器用 cont 表示、组件用 comp 表示，书写代码的方式是_____。
 A. comp.add(BorderLayout.CENTER,cont);
 B. comp.add(cont,BorderLayout.CENTER);
 C. cont.add(BorderLayout.CENTER,comp);
 D. cont.add(comp,BorderLayout.CENTER);
5. 窗体和面板容器的默认布局分别是_____。
 A. 边界布局、流布局　　B. 流布局、边界布局
 C. 边界布局、卡片布局　　D. 卡片布局、空布局
6. 下面不是容器的组件是_____。
 A. JPanel　　B. JFrame　　C. JScrollPane　　D. JList
7. 下面代码中，设置容器的布局为空的正确语句是_____。
 A. set(null)　　B. set(NULL)　　C. setLayout(null)　　D. setLayout(NULL)
8. JButton 的父类是____。
 A. Container　　B. JComponent　　C. Button　　D. AbstractButton
9. 编写一个包含有一个文本区域和 3 个按钮的界面程序，如图 3-15 所示。



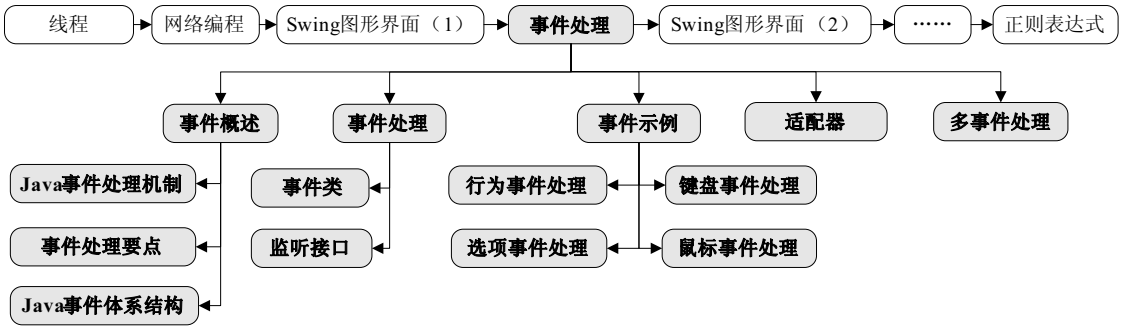
图 3-15　要实现的界面

第 4 章　事件处理

本章目标

- 理解 Java 事件处理机制
- 掌握事件控制的要点
- 掌握 AWT 事件结构
- 掌握基本事件控制
- 掌握常用的事件类、监听接口和适配器
- 掌握事件处理的步骤
- 掌握行为、键盘和鼠标事件的处理
- 掌握 Java 的多事件处理机制

学习导航



任务描述

【描述 4.D.1】

单击按钮，在文本栏中显示单击次数。

【描述 4.D.2】

两个组合框，一个显示省份，一个显示城市，当省份改变时，城市也改变（联动效果）。

【描述 4.D.3】

使用键盘方向键控制按钮走动。

【描述 4.D.4】

实现一个画板程序，当用户在面板上拖动鼠标时，显示其轨迹。

【描述 4.D.5】

通过实现“小猫抓不到老鼠”示例，演示适配器的使用。

【描述 4.D.6】

实现计算器，当用户单击按钮或敲击键盘时都能进行运算。

4.1　事件概述

对于图形用户界面的应用程序，要想实现用户界面的交互，必须通过事件处理。事件处理就是在事件驱动机制中，应用程序可以响应事件来执行一系列的操作。事件驱动机制是在图形界面应用程序中由于用户操作（如单击鼠标或按键盘某个键），程序代码或系统内部而产生“事件”。这种基于事件驱动机制的事件处理是目前实现与用户交互的最好方式。本章讲解在 Java 中如何实现事件处理。

4.1.1　Java 事件处理机制

在图形用户界面中，当用户使用鼠标单击一个按钮、在列表框中进行选择，或者单击窗口右上角的“x”（关闭按钮）等时，这些用户操作的动作就会触发一个相应的事件。事件发生后，将被相应的事件监听器监听到，事件监听器捕获事件并调用相应的方法进行处理。这种处理机制其实是一种委派式的事件处理方式：组件将事件处理委托给特定的事件处理对象，当该组件发生指定的事件时，就通知所委托的事件处理对象，由此对象来处理这个事件。这个受委托处理事件的对象称为事件监听对象。每个组件均可以针对特定的事件指定一个或多个事件监听对象，由这些事件监听对象负责监听并处理事件。

Java 的事件处理机制如图 4-1 所示。

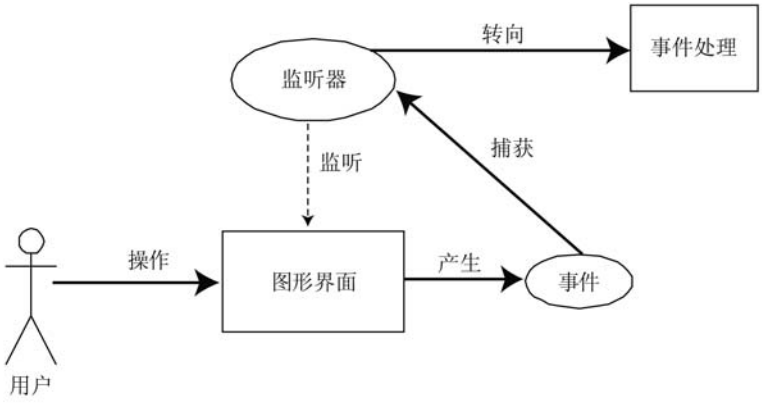


图 4–1　Java 的事件处理机制

4.1.2　事件处理要点

Java 中进行事件处理的步骤如下：

01 创建监听类，在事件处理方法中编写事件处理代码。
02 创建监听对象。
03 利用组件的 addXXXListener()方法将监听对象注册到组件上。

这里需要注意监听类、事件处理方法和监听对象这 3 个概念的区别与联系：

- **监听类**：是一个扩展监听接口的类，它可以扩展一个或多个监听接口。监听接口是 Java 类库中已定义好的接口，其详细信息见后续章节。
- **事件处理方法**：是监听接口中已经定义好的相应的事件处理方法。在创建监听类时，需要重写这些事件处理方法，将事件处理的代码放入相应的方法中。
- **监听对象**：是监听类的一个实例对象，具有监听功能。这样，将此监听对象注册到组件上，当该组件上发生相应的事件时，将会被此监听对象捕获并调用相应的方法进行处理。

4.1.3 Java 事件体系结构

在 Java 事件体系结构中，以 3 种对象为中心来组成一个完整的事件模型：

- **事件源**：是产生或抛出事件的对象。事件源通常是 GUI 组件。
- **事件**：是一个描述事件源状态改变的对象。事件不是通过 new 运算符创建的，而是由用户操作触发的。
- **监听接口**：在监听接口中定义了抽象的事件处理方法，监听类中需实现监听接口中提供的方法。

4.2 事件处理

4.2.1 事件类

事件类用于封装事件处理所必需的基本信息，如事件源、事件信息等。所有的事件类都继承自 EventObject，Java 将所有组件可能发生的事件进行分类，具有共同特征的事件被抽象为一个事件类 AWTEvent。AWTEvent 是所有 AWT 事件的根事件类，此类及其子类取代了原来的 java.awt.Event 类，其常用方法及功能见表 4-1。

表 4–1　AWTEvent 的常用方法及功能

方法	功能说明
int getID()	返回事件类型
String paramString()	返回此 Event 状态的字符串，此方法仅在进行调试的时候使用
Object getSource()	从 EventObject 类继承的方法，用于返回事件源的对象

常用的事件类包括 ActionEvent（动作事件）、MouseEvent（鼠标事件）、KeyEvent（键盘事件）等，其继承层次关系如图 4-2 所示。

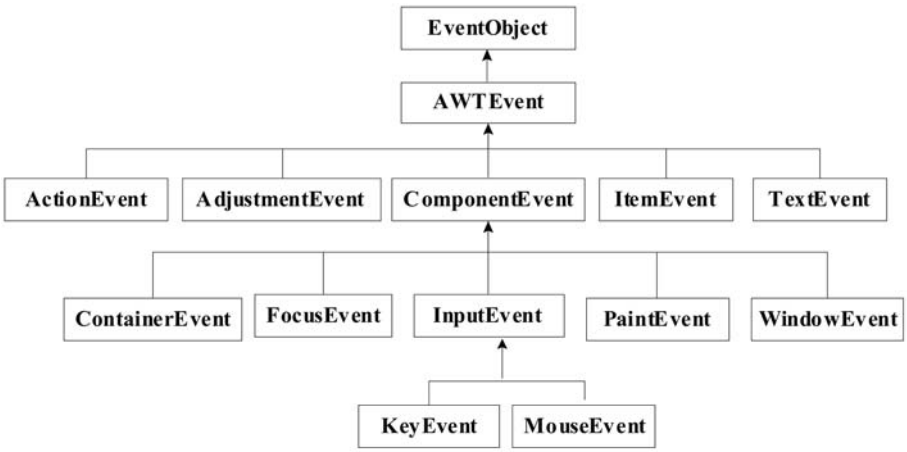


图 4–2　事件类

注意　许多事件类是 Swing 组件和 AWT 组件（最早的图形组件，现已不提倡使用）所共有的，这样的事件类都在 java.awt.event 包中。javax.swing.event 包声明了 Swing 组件所特有的事件类。

表 4-2 列出了各事件类及其功能说明，以及可以产生此类事件的组件，即事件源。

表 4–2　事件类及其功能

事件类	说明	事件源
ActionEvent	行为事件，最常用的一个事件	按钮、列表、菜单项等
AdjustmentEvent	调节事件，当调节滚动条时会生成此事件	滚动条
ItemEvent	选项事件，当选择不同的选项时会生成此事件	列表、组合框等
FocusEvent	焦点事件，当组件获得或失去焦点时会生成此事件	能接受焦点的组件
KeyEvent	键盘事件，当敲击键盘时会生成此事件	能接受焦点的组件
MouseEvent	鼠标事件，当操作鼠标时会生成此事件	所有组件
WindowEvent	窗口事件，当窗口状态改变时会生成此事件	窗体

4.2.2　监听接口

监听接口中定义了抽象的事件处理方法，这些方法针对不同的操作进行不同的处理。在程序中，通常使用监听类实现监听接口中的事件处理方法。监听接口定义在 java.awt.event 包中，该包中提供了不同事件的监听接口，这些接口中定义了不同的抽象的事件处理方法。表 4-3 列出了常用的监听接口及其事件处理方法。

表 4–3　监听接口及其事件处理方法

监听接口	处理方法	功能说明
ActionListener	actionPerformed(ActionEvent e)	行为处理

（续表）

监听接口	处理方法	功能说明
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)	调节值改变
ItemListener	itemStateChanged(ItemEvent e)	选项值状态改变
FocusListener	focusGained(FocusEvent e)	获得聚焦
	focusLost(FocusEvent e)	失去聚焦
KeyListener	keyPressed(KeyEvent e)	按下键盘
	keyReleased(KeyEvent e)	松开键盘
	keyTyped(KeyEvent e)	敲击键盘
MouseListener	mouseClicked(MouseEvent e)	鼠标点击
	mouseEntered(MouseEvent e)	鼠标进入
	mouseExited(MouseEvent e)	鼠标退出
	mousePressed(MouseEvent e)	鼠标按下
	mouseReleased(MouseEvent e)	鼠标松开
MouseMotionListener	mouseDragged(MouseEvent e)	鼠标拖动
	mouseMoved(MouseEvent e)	鼠标移动
WindowListener	windowActivated(WindowEvent e)	窗体激活
	windowClosed(WindowEvent e)	窗体关闭以后
	windowClosing(WindowEvent e)	窗体正在关闭
	windowDeactivated(WindowEvent e)	窗体失去激活
	windowDeiconified(WindowEvent e)	窗体非最小化
	windowIconified(WindowEvent e)	窗体最小化
	windowOpened(WindowEvent e)	窗体打开后

4.3 事件示例

4.3.1 行为事件处理示例

下述代码用于实现任务描述 4.D.1，在文本框中显示单击按钮的次数，代码如下：

【描述 4.D.1】ClickCount.java

```
public class ClickCount extends JFrame {
    JPanel panel;
    JButton button;
    JTextField field;
```

```
    // 记录单击次数
    int count = 0;
    public ClickCount() {
        super("行为事件测试");
        panel = new JPanel();
        button = new JButton("点击");
        field = new JTextField(10);
        panel.add(button);
        panel.add(field);
        // 创建一个监听对象
        ButtonListener bl = new ButtonListener();
        // 注册监听
        button.addActionListener(bl);
        this.add(panel);
        this.setSize(200, 100);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        ClickCount f = new ClickCount();
        f.setVisible(true);
    }
    // 创建扩展 ActionListener 的监听类
    public class ButtonListener implements ActionListener {
        // 重写 ActionListener 接口中的事件处理方法 actionPerformed()
        public void actionPerformed(ActionEvent e) {
            count++;
            // 在文本框中显示次数
            field.setText(count + "次");
        }
    }
}
```

在上述代码中，ClickCount 类内创建了一个扩展 ActionListener 的监听类 ButtonListener。ButtonListener 类是一个内部类，内部类可直接访问其所在类中的其他成员。在 ClickCount 类的构造方法中实例化一个 ButtonListener 类的对象 bl，该对象就是监听对象；然后将此监听对象用 addActionListener()方法注册到按钮上，这样此时的监听对象监视着按钮是否被单击，如果按钮被单击，则调用其 actionPerformed()方法进行处理。处理过程如图 4-3 所示。

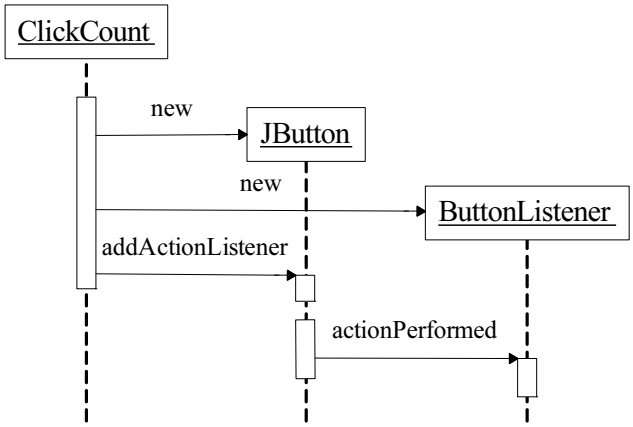


图 4-3　事件通告

执行结果如图 4-4 所示。



图 4-4　ClickCount 执行结果

4.3.2　选项事件处理示例

下述代码用于实现任务描述 4.D.2，实现省份组合框的联动效果。代码如下：

【描述 4.D.2】ItemDemo.java

```
public class ItemDemo extends JFrame implements ItemListener {
    private JPanel p;
    private JComboBox cmbProv, cmbCity;
    public ItemDemo() {
        super("组合框联动");
        p = new JPanel();
        String str[] = { "山东", "安徽", "江苏" };
        cmbProv = new JComboBox(str);
        cmbCity = new JComboBox();
        p.add(cmbProv);
        p.add(cmbCity);
        // 在省份组合框中注册监听对象
        cmbProv.addItemListener(this);
        this.add(p);
        this.setSize(200, 150);
```

```
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 重写 ItemListener 中的事件处理方法 itemStateChanged()
    public void itemStateChanged(ItemEvent e) {
        // 当选项被选中时
        if (e.getStateChange() == ItemEvent.SELECTED) {
            // 清空城市组合框中的所有选项
            cmbCity.removeAllItems();
            // 获取选中的选项值（省份）
            String item = e.getItem().toString();
            if (item.equals("山东")) {
                cmbCity.addItem("济南");
                cmbCity.addItem("青岛");
                cmbCity.addItem("烟台");
            }
            if (item.equals("安徽")) {
                cmbCity.addItem("合肥");
                cmbCity.addItem("芜湖");
                cmbCity.addItem("蚌埠");
            }
            if (item.equals("江苏")) {
                cmbCity.addItem("南京");
                cmbCity.addItem("苏州");
                cmbCity.addItem("扬州");
            }
        }
    }
    public static void main(String[] args) {
        ItemDemo f = new ItemDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，ItemDemo 类本身扩展了 ItemListener 接口，即其本身（this）就是一个监听对象，因此在其构造方法中直接将 this 对象注册到组合框中即可，不必再新建监听对象。itemStateChanged()方法是 ItemListener 接口中的事件处理方法，用来处理 ItemEvent 事件。ItemEvent 事件的 getStateChange()方法用来获取选项状态的改变，其返回值为：SELECTED（选中）或 DESELECTED（不被选中）。

执行结果如下图 4-5 所示。



图 4-5　ItemDemo 执行结果

4.3.3　键盘事件处理

下述代码用于实现任务描述 4.D.3，用键盘方向键控制按钮走动。代码如下所示：

【描述 4.D.3】KeyDemo.java

```
public class KeyDemo extends JFrame implements KeyListener {
    private JPanel p;
    private JButton b;
    public KeyDemo() {
        super("键盘控制");
        p = new JPanel();
        b = new JButton("用方向键控制");
        p.add(b);
        // 注册监听
        b.addKeyListener(this);
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 敲击键盘的事件处理方法
    public void keyTyped(KeyEvent e) {
    }
    // 键盘按下的事件处理方法
    public void keyPressed(KeyEvent e) {
        // 获取按下键盘的码值
        int key = e.getKeyCode();
        // 获得按钮当前的 x,y 轴坐标
        int x = b.getX();
        int y = b.getY();
        if (key == KeyEvent.VK_RIGHT) {
            // 向右，x 轴坐标值增加
            b.setLocation(x + 5, y);
        } else if (key == KeyEvent.VK_LEFT) {
            // 向左，x 轴坐标值减少
```

```
            b.setLocation(x - 5, y);
        } else if (key == KeyEvent.VK_UP) {
            // 向上，y 轴坐标值减少
            b.setLocation(x, y - 5);
        } else if (key == KeyEvent.VK_DOWN) {
            // 向下，y 轴坐标值增加
            b.setLocation(x, y + 5);
        }
    }
    // 键盘松开的事件处理方法
    public void keyReleased(KeyEvent e) {
    }
    public static void main(String[] args) {
        KeyDemo f = new KeyDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，KeyDemo 类扩展了 KeyListener 接口，该接口中有 3 个事件处理方法，分别为：keyTyped()、keyPressed()和 keyReleased()。在 keyPressed()方法中加入处理代码，其他两个方法虽无处理代码，但必须保留。因为扩展接口时，类必须重写接口中的所有方法，哪怕有些方法不需要，也必须重写。

键盘事件 KeyEvent 类中对每个键盘按键都定义了一个常量与之对应，VK_RIGHT 代表方向键“→”，VK_LEFT 代表“←”，VK_UP 代表“↑”，VK_DOWN 代表“↓”。

KeyEvent 类中常用的两个方法如下：

- int getKeyCode()：获取键盘按键码值。
- char getKeyChar()：获取键盘按键上的字符。

执行结果如图 4-6 所示。



图 4–6　KeyDemo 执行结果

操作键盘的方向键，观察按钮位置的变化。

4.3.4 鼠标事件处理

下述代码用于实现任务描述 4.D.4，拖动鼠标在面板中画图。代码如下所示：

【描述 4.D.4】MouseDemo.java

```
public class MouseDemo extends JFrame implements MouseMotionListener {
    private JPanel p;
    int x, y;
    public MouseDemo() {
        super("画板");
        p = new JPanel();
        // 注册监听
        p.addMouseMotionListener(this);
        this.add(p);
        this.setSize(400,300);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 重写 JFrame 的 paint()方法，此方法用于在窗体中画图
    public void paint(Graphics g) {
        // 设置画笔的颜色
        g.setColor(Color.blue);
        // 画一个实心圆
        g.fillOval(x, y, 10, 10);
    }
    // 鼠标移动的处理方法
    public void mouseMoved(MouseEvent e) {
    }
    // 鼠标拖动的处理方法
    public void mouseDragged(MouseEvent e) {
        // 获取鼠标当前的坐标
        x = e.getX();
        y = e.getY();
        // 重画，repaint()触发 paint()
        this.repaint();
    }
    public static void main(String[] args) {
        MouseDemo f = new MouseDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，除了重写 MouseMotionListener 接口中的鼠标事件处理方法外，还需要重写窗体的 paint()方法，此方法用于在窗体中画图。repaint()可以触发 paint()方法，进行重画。

执行结果如图 4-7 所示。

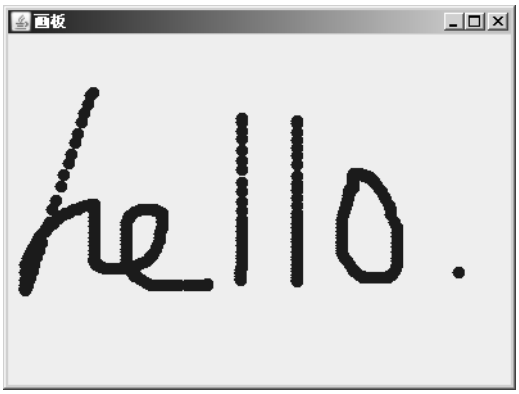


图 4–7　MouseDemo 运行结果

4.4　适配器

在任务描述 4.D.3 和 4.D.4 中，扩展监听接口时，有些事件处理方法是不需要的，但必须重写。出于简化代码的目的，java.awt.event 包中又提供了一套抽象适配器类，分别实现每个具有多个事件处理方法的监听接口。这样继承适配器后，可以仅重写需要的事件处理方法。

适配器类见表 4-4。

表 4–4　适配器类列表

适配器类	实现接口
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

下述代码用于实现任务描述 4.D.5，使用鼠标适配器（MouseAdapter）实现“小猫抓不到老鼠”的特效。

【描述 4.D.5】MouseAdapterDemo.java

```
//继承 MouseAdapter 适配器类
public class MouseAdapterDemo extends MouseAdapter {
    JFrame f;
    JPanel p;
    JButton b;
```

```
    public MouseAdapterDemo() {
        f = new JFrame("抓不到");
        p = new JPanel();
        b = new JButton("你就碰不到我");
        b.addMouseListener(this);
        p.add(b);
        f.add(p);
        f.setSize(400,300);
        f.setLocation(100, 100);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
    // 重写鼠标进入的事件处理方法
    public void mouseEntered(MouseEvent e) {
        // 产生随机数，作为新坐标
        int x = (int) (Math.random() * 400);
        int y = (int) (Math.random() * 300);
        b.setLocation(x, y);
    }
    public static void main(String args[]) {
        new MouseAdapterDemo();
    }
}
```

在上述代码中，使用 MouseAdapter 而不使用 MouseListener，这样只需重写需要的事件处理方法 mouseEntered()方法，MouseListener 接口中的其他 4 个鼠标事件处理方法就不必重写了，如此使用适配器简化了程序代码。

执行结果如图 4-8 所示。



图 4-8　MouseAdapterDemo 执行结果

4.5　多事件处理

到目前为止，示例代码中都是处理一种事件，下面讲述程序的多种事件处理。以计算器为例，我们不仅要对单击按钮的行为事件进行处理，还要对敲击键盘的键盘事件进行处理。

下述代码用于实现任务描述 4.D.6，实现计算器，当用户单击按钮或敲击键盘都能进行运算。

【描述 4.D.6】Calculator.java

```
public class Calculator extends JFrame {
    // 声明一个文本栏控件，用于显示计算结果
    private JTextField txtResult;
    private JPanel p;
    // 定义一个字符串数组，将计算器中按钮的文字都放在该数组中
    private String name[] = { "7", "8", "9", "+", "4", "5", "6", "-", "1", "2",
            "3", "*", "0", ".", "=", "/" };
    // 声明一个按钮数组，该数组的长度以字符串数组的长度为准
    private JButton button[] = new JButton[name.length];
    // 定义一个存放计算结果的变量，初始为 0
    private double result = 0;
    // 存放最后一个操作符，初始为"="
    private String lastCommand = "=";
    // 标识是否是开始
    private boolean start = true;
    public Calculator() {
        super("计算器");
        // 实例化文本栏控件
        txtResult = new JTextField(20);
        //设置文本框不是焦点状态
        txtResult.setFocusable(false);
        // 将文本栏控件放置在窗体框架的上方（北部）
        this.add(txtResult, BorderLayout.NORTH);
        // 实例化面板对象，同时设置此面板布局为 4 行 4 列的网格布局
        p = new JPanel(new GridLayout(4, 4));
        // 循环实例化按钮对象数组
        // 实例化按钮监听对象
        ButtonAction ba = new ButtonAction();
        // 实例化键盘监听对象
        KeyAction ka = new KeyAction();
        for (int i = 0; i < button.length; i++) {
            button[i] = new JButton(name[i]);
            // 注册监听
            button[i].addActionListener(ba);
```

```
            button[i].addKeyListener(ka);
            p.add(button[i]);
        }
        this.add(p, BorderLayout.CENTER);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 计算
    public void calculate(double x) {
        if (lastCommand.equals("+"))
            result += x;
        else if (lastCommand.equals("-"))
            result -= x;
        else if (lastCommand.equals("*"))
            result *= x;
        else if (lastCommand.equals("/"))
            result /= x;
        else if (lastCommand.equals("="))
            result = x;
        // 将结果显示在文本栏
        txtResult.setText("" + result);
    }
    // 单击按钮监听
    private class ButtonAction implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String input = e.getActionCommand();
            // 单击操作符号按钮
            if (input.equals("+") || input.equals("-") || input.equals("*")
                    || input.equals("/") || input.equals("=")) {
                if (start) {
                    if (input.equals("-")) {
                        txtResult.setText(input);
                        start = false;
                    } else
                        lastCommand = input;
                } else {
                    calculate(Double.parseDouble(txtResult.getText()));
                    lastCommand = input;
                    start = true;
                }
            } else {
                if (start) {
```

```
                    txtResult.setText("");
                    start = false;
                }
                txtResult.setText(txtResult.getText() + input);
            }
        }
    }
    // 键盘监听
    private class KeyAction extends KeyAdapter {
        public void keyTyped(KeyEvent e) {
            char key = e.getKeyChar();
            // 敲击的键盘是数字
            if (key == '0' || key == '1' || key == '2' || key == '3'
                    || key == '4' || key == '5' || key == '6' || key == '7'
                    || key == '8' || key == '9' || key == '9') {
                if (start) {
                    txtResult.setText("");
                    start = false;
                }
                txtResult.setText(txtResult.getText() + key);
            }
            // 敲击的键盘是操作符号
            else if (key == '+' || key == '-' || key == '*' || key == '/'
                    || key == '=') {
                if (start) {
                    if (key == '-') {
                        txtResult.setText(String.valueOf(key));
                        start = false;
                    } else
                        lastCommand = String.valueOf(key);
                } else {
                    calculate(Double.parseDouble(txtResult.getText()));
                    lastCommand = String.valueOf(key);
                    start = true;
                }
            }
        }
    }
    public static void main(String[] args) {
        Calculator f = new Calculator();
        f.setVisible(true);
    }

}
```

在上述代码中，创建了两个监听类，分别监听单击按钮的行为事件和敲击键盘的事件。进行事件处理时数字和操作符号分别处理，其中“-”有两种意思：“负”和“减”，当在算式开始时代表负号，否则是减法符号。

执行结果如图 4-9 所示。



图 4–9　计算器运行结果

小结

通过本章的学习，学生应该能够学会：

- 事件用于实现用户界面的交互，是 GUI 编程的重要组成部分。
- Java 最新的事件处理方法是基于委派事件模型的。
- 事件包含 3 个组件：事件对象、事件源和事件处理程序。
- 事件是一个描述事件源状态改变的对象，事件不是通过 new 运算符创建的，而是由用户操作触发的。
- 一个事件源可能会生成不同类型的事件。
- 可以将同一事件发送到多个监听器对象。
- 监听类是实现监听接口的类，监听对象是监听类的实例化对象。
- java.awt.event 包中常用的事件类：ActionEvent、AdjustmentEvent、ItemEvent、FocusEvent、KeyEvent、MouseEvent、WindowEvent。
- java.awt.event 包中常用的监听接口：ActionListener、AdjustmentListener、ItemListener、FocusListener、KeyListener、MouseListener、MouseMotionListener、WindowListener。
- 在组件中注册监听器的方法是：addXXXListener()。
- 适配器是实现相关监听接口的类，目的是简化程序代码。

练习

1. 事件监听接口中的方法的返回值是_____。

 A. int　　B. String　　C. void　　D. Object

2. 在 Java 中，要处理 Button 类对象的事件，以下_____是可以处理这个事件的接口。

 A. FocusListener　　B. ComponentListener

 C. WindowListener　　D. ActionListener

3. 要判断关闭窗口的事件，应该添加_____监听器。

 A. 鼠标监听器　　B. 鼠标移动监听器

 C. 窗口监听器　　D. 以上监听器均可

4. 下述代码中，如果单击“TEST”按钮，标准输出的消息是_____。

```
public class Exercise extends JFrame{
    public Exercise(){
        super("事件测试");
        Button b=new Button("TEST");
        b.addMouseListener(new Tester());
        this.add(b);
        this.setSize(200, 150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
}
class Tester implements MouseListener{
    public void actionPerformed(ActionEvent e){
            System.out.println("按钮发生动作");
        }
        public void mouseClicked(MouseEvent e){
            System.out.println("按钮被单击");
        }
        public  void mousePressed(MouseEvent e){}
        public  void mouseReleased(MouseEvent e){}
        public  void mouseEntered(MouseEvent e){}
        public  void mouseExited(MouseEvent e){}
}
```

 A. “按钮发生动作”　　B. “按钮被单击”

 C. “按钮发生动作”和“按钮被单击”　　D. 以上都不对

5. 下面_____方法用于获取事件源。

 A. getEvent()　　B. getCommond()

 C. getText()　　D. getSource()

6. 简述适配器和监听接口的区别。

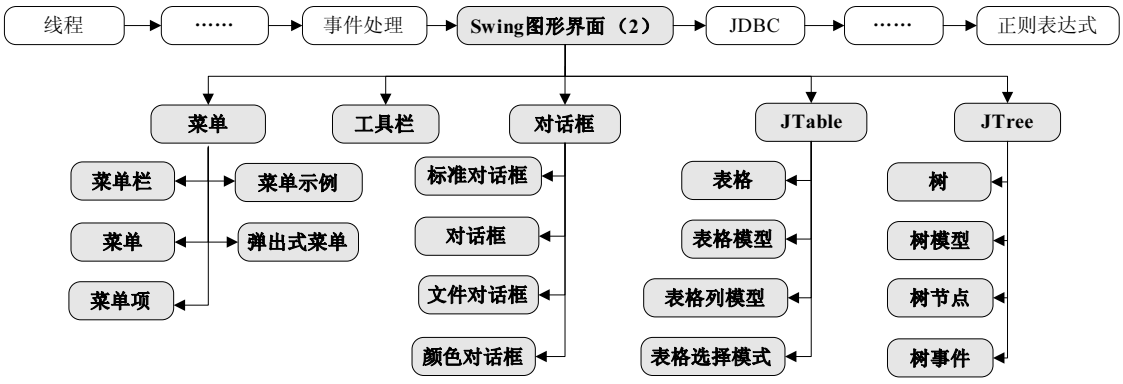
7. 编写一个程序：界面中有两个按钮（“上一张”和“下一张”），当单击“上一张”按钮时显示上一张图片，当单击“下一张”按钮时显示下一张图片。

第 5 章　Swing 图形界面（2）

本章目标

- 掌握菜单的创建及事件处理
- 掌握工具栏的创建
- 理解对话框的种类及创建
- 学会使用文件对话框打开和保存文件
- 了解使用 JDialog 创建自己的对话框的方法
- 掌握 JTable 组件的使用方式
- 掌握 JTree 组件的使用方式

学习导航



任务描述

【描述 5.D.1】

创建一个具有三级菜单的界面。

【描述 5.D.2】

在一个窗体中有 4 个按钮，单击 4 个按钮显示不同的标准对话框。

【描述 5.D.3】

创建一个窗体界面，“用户管理”菜单中提供一个“添加新用户”的菜单，单击此菜单，弹出一个模式对话框，输入新用户信息（用户名、密码和权限），验证信息（用户名和密码不能为空，密码长度 6~10），当信息符合要求则返回并显示到窗体的文本域中。

【描述 5.D.4】

实现一个记事本应用程序，打开文件或保存文件。

【描述 5.D.5】

创建一个 JTabel，提供 3 个改变表格选中模式的按钮，当选择不同按钮时，表格的选择模式就不同。

【描述 5.D.6】

创建一个窗口界面，左边显示树目录，右边是文本域，当选择不同的树节点时，在文本域中显示节点的内容文本。

5.1 菜单

菜单是 GUI 图形界面的重要组成部分，菜单通过减少用户可以同时看到的组件的数目来简化 GUI。菜单通常是由菜单栏（JMenuBar）、菜单（JMenu）和菜单项（JMenuItem）来组合实现的，通常菜单栏中可以添加一个或多个菜单，每个菜单中又可以添加一个或多个菜单项，每个菜单项代表允许用户选择的某些功能。

注意 菜单可以嵌套，即一个菜单中不仅可以添加菜单项，也可以添加另外一个菜单对象。菜单的嵌套通常称为“多级菜单”。

5.1.1 菜单栏（JMenuBar）

JMenuBar 类用于创建一个菜单栏对象，该组件显示一个顶级菜单选项列表，每个选项都连接着下拉菜单。菜单栏是一个可以添加到任何位置的组件，在正常情况下，显示在框架的顶部。

菜单栏对象可以通过 JFrame 类的 setJMenuBar()方法将其添加到框架的顶部，例如：

```
frame.setJMenuBar(menuBar);
```

注意 在 Swing GUI 中，菜单只能添加到提供了 setJMenuBar()方法的类对象中。JFrame 和 JDialog 就是这样的两个类。

5.1.2 菜单（JMenu）

JMenu 类用于创建一个菜单对象，菜单对象中可以添加菜单项、分隔线和子菜单，组成一个下拉列表形式的菜单。JMenu 类中的常用方法及功能见表 5-1。

表 5-1 JMenu 类的常用方法及功能

方法名	功能说明
JMenu()	构造方法，用于创建一个新的、无文本的菜单对象
JMenu(String str)	构造方法，用于创建一个新的、指定文本的菜单对象
JMenu(String str, boolean bool)	构造方法，用于创建一个新的、指定文本的、是否分离式的菜单对象
add(Component c)	将组件添加到菜单的末尾
addSeparator()	将分隔线追加到菜单的末尾

5.1.3 菜单项（JMenuItem）

JMenuItem 菜单选项类用于创建一个菜单选项对象，其本质上是位于列表中的按钮。当用户选择菜单项时，则执行与菜单项关联的操作。JMenuItem 类中的常用方法及功能见表 5-2。

表 5-2　JMenuItem 类中的常用方法及功能

方法名	功能说明
JMenuItem ()	构造方法，用于创建一个新的、无文本和图标的菜单对象
JMenuItem(Icon icon)	构造方法，用于一个新的、指定图标的菜单对象
JMenuItem(String text)	构造方法，用于创建一个新的、指定文本的菜单对象
JMenuItem(String text, Icon icon)	构造方法，用于创建一个新的、指定文本和指定图标的菜单对象，菜单项对象可以添加到菜单对象中
addActionListener(ActionListener l)	从 AbstractButton 类中继承的方法，将监听对象添加到菜单项中

5.1.4　菜单示例

创建菜单的步骤如下：

01 创建一个 JMenuBar 菜单栏对象，将其设置到窗体中。

02 创建若干个 JMenu 菜单对象，将其放置到 JMenuBar 对象中，或按要求放到其他 JMenu 对象中。

03 创建若干个 JMenuItem 菜单项对象，将其放置到相应的 JMenu 对象中。

下述代码用于实现任务描述 5.D.1，实现一个三级菜单。

【描述 5.D.1】MenuDemo.java

```
public class MenuDemo extends JFrame {
    private JPanel p;
    // 声明菜单栏
    private JMenuBar menuBar;
    // 声明菜单
    private JMenu menuFile, menuEdit, menuHelp, menuNew;
    // 声明菜单选项
    private JMenuItem miSave, miExit, miCopy, miPost, miAbout, miC, miJava,
            miEmpty;
    public MenuDemo() {
        super("菜单");
        p = new JPanel();
        // 创建菜单栏对象
        menuBar = new JMenuBar();
        // 将菜单栏设置到窗体中
        this.setJMenuBar(menuBar);
        // 创建菜单
        menuFile = new JMenu("文件");
        menuEdit = new JMenu("编辑");
        menuHelp = new JMenu("帮助");
```

```
        menuNew = new JMenu("新建");
        // 将菜单添加到菜单栏
        menuBar.add(menuFile);
        menuBar.add(menuEdit);
        menuBar.add(menuHelp);
        // 将新建菜单添加到文件菜单中
        menuFile.add(menuNew);
        // 在菜单中添加分隔线
        menuFile.addSeparator();
        // 创建菜单选项
        miSave = new JMenuItem("保存");
        miExit = new JMenuItem("退出");
        miCopy = new JMenuItem("复制");
        miPost = new JMenuItem("粘贴");
        miAbout = new JMenuItem("关于");
        miC = new JMenuItem("C++");
        miJava = new JMenuItem("JAVA");
        miEmpty = new JMenuItem("空白文档");
        // 将菜单项添加到菜单中
        menuFile.add(miSave);
        menuFile.add(miExit);
        menuEdit.add(miCopy);
        menuEdit.add(miPost);
        menuHelp.add(miAbout);
        menuNew.add(miC);
        menuNew.add(miJava);
        menuNew.add(miEmpty);
        this.add(p);
        this.setSize(300, 250);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        MenuDemo f = new MenuDemo();
        f.setVisible(true);
    }
}
```

在上述代码中使用 setJMenuBar()方法将菜单栏设置到窗体顶部，使用 add()方法添加菜单和菜单项。

执行结果如图 5-1 所示。



图 5–1　MenuDemo 运行结果

5.1.5　弹出式菜单

JPopupMenu 弹出式菜单是一种不固定在菜单栏中、可以随处浮动的菜单。建立浮动菜单与建立一般菜单很相似，但是弹出菜单没有标题。JPopupMenu 类中的常用方法及功能见表 5-3。

表 5–3　JPopupMenu 类中的常用方法及功能

方法名	功能说明
JPopupMenu()	构造方法，用于创建一个新的、无文本的菜单对象
JPopupMenu(String label)	构造方法，用于创建一个新的、指定文本的菜单对象
add(Component c)	将组件添加到菜单末尾
addSeparator()	将分隔线追加到菜单的末尾

【代码 5–1】PopupMenuDemo.java

```
public class PopupMenuDemo extends JFrame {
    private JPanel p;
    // 声明弹出菜单
    private JPopupMenu popMenu;
    // 声明菜单选项
    private JMenuItem miUndo, miCopy, miPost, miCut;
    public PopupMenuDemo() {
        super("弹出菜单");
        p = new JPanel();
        // 创建弹出菜单对象
        popMenu = new JPopupMenu("弹出");
        // 创建菜单选项
        miUndo = new JMenuItem("撤销");
        miCopy = new JMenuItem("复制");
        miPost = new JMenuItem("粘贴");
        miCut = new JMenuItem("剪切");
        // 将菜单选项添加到菜单中
        popMenu.add(miUndo);
        popMenu.addSeparator();
```

```
        popMenu.add(miCopy);
        popMenu.add(miPost);
        popMenu.add(miCut);
        // 注册监听
        p.addMouseListener(new MouseAction());
        this.add(p);
        this.setSize(300, 200);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        PopupMenuDemo f = new PopupMenuDemo();
        f.setVisible(true);
    }
    // 鼠标监听类
    private class MouseAction extends MouseAdapter {
        // 重写鼠标单击事件处理方法
        public void mouseClicked(MouseEvent e) {
            // 如果单击鼠标左键
            if (e.getButton() == MouseEvent.BUTTON1) {
                int x = e.getX();
                int y = e.getY();
                // 在面板中鼠标所在位置显示弹出菜单
                popMenu.show(p, x, y);
            }
        }
    }
}
```

上述代码中使用 JPopupMenu 创建弹出菜单，再在弹出菜单中添加菜单选项。使用鼠标监听，当鼠标在面板中单击左键时显示弹出菜单。JPopupMenu 中的 show()方法用于显示弹出菜单。

执行结果如图 5-2 所示。



图 5-2　PopupMenuDemo 运行结果

5.2　工具栏

工具栏是在程序中提供快速访问常用命令的按钮栏。在 Swing 中，用于实现工具栏功能的类是 JToolBar，其常用的方法及功能见表 5-4。

表 5–4　JToolBar 类常用的方法及功能

方法名	功能说明
JToolBar()	构造方法，用于创建新的工具栏，默认的方向为 HORIZONTAL（水平）
JToolBar(int orientation)	构造方法，用于创建一个具有指定方向的新工具栏
JToolBar(String name)	构造方法，用于创建一个具有指定 name 的新工具栏
JToolBar(String name, int orientation)	构造方法，用于创建一个指定 name 和方向的新工具栏
addSeparator()	将分隔线添加到工具栏的末尾
setMargin(Insets m)	设置工具栏边框和它的按钮之间的空白

示例代码如下所示：

【代码 5–2】ToolBarDemo.java

```
public class ToolBarDemo extends JFrame {
    private JPanel p;
    //声明工具栏
    private JToolBar toolBar;
    private JButton btnCopy,btnPost,btnCut;
    public ToolBarDemo() {
        super("工具栏");
        p = new JPanel();
        //创建工具栏
        toolBar=new JToolBar();
        //将工具栏对象添加到窗体的上方（北面）
        this.add(toolBar,BorderLayout.NORTH);
        //创建按钮对象，按钮上有图片
        btnCopy=new JButton(new ImageIcon("D:\\copy.gif"));
        btnPost=new JButton(new ImageIcon("D:\\post.gif"));
        btnCut=new JButton(new ImageIcon("D:\\cut.gif"));
        //设置按钮的工具提示文本
        btnCopy.setToolTipText("复制");
        btnPost.setToolTipText("粘贴");
        btnCut.setToolTipText("剪切");
        //将按钮添加到工具栏中
        toolBar.add(btnCopy);
        toolBar.add(btnPost);
        toolBar.add(btnCut);
```

```
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        ToolBarDemo f = new ToolBarDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 JToolBar 创建工具栏，在工具栏中可以加入按钮控件。工具栏的特殊之处在于可以把它移动到任何地方，可以将它拖曳到窗体的 4 个边上或脱离出来。

运行结果如图 5-3 所示。



图 5–3　ToolBarDemo 执行结果

5.3　对话框

5.3.1　标准对话框

GUI 中包含了一组标准的对话框，用于显示消息或获取信息。在 Swing 中，这些标准对话框包含在 JOptionPane 组件中。JOptionPane 类中提供了 4 种用于显示不同对话框的静态方法，方法说明如表 5-5 所示。

表 5–5　JOptionPane 中的 4 种方法

方法名	功能说明
showConfirmDialog	显示确认对话框，等待用户确认（OK/Cancel）
showInputDialog	显示输入对话框，等待用户输入信息。以字符串形式返回用户输入的信息
showMessageDialog	显示消息对话框，等待用户单击“OK”按钮
showOptionDialog	显示选择对话框，等待用户在一组选项中选择。将用户选择的选项下标值返回

1.　消息对话框

JOptionPane 中用于显示消息对话框的方法如下：

- void showMessageDialog(Component parentComponent,Object message)：显示一个标题是“Message”的、指定信息的消息对话框。

- void showMessageDialog(Component parentComponent,Object message,String title,int messageType)：显示一个指定信息、标题和消息类型的消息对话框。

参数说明：

- parentComponent：父组件。如果为 null，对话框将显示在屏幕中央；否则根据父组件所在窗体来确定位置。
- message：信息内容。
- title：标题。
- messageType：消息类型。在对话框中，左边显示的图标取决于消息类型，不同的消息类型显示不同的图标。在 JOptionPane 中提供了五种消息类型：ERROR_MESSAGE（错误）、INFORMATION_MESSAGE（通知）、WARNING_MESSAGE（警告）、QUESTION_MESSAGE（疑问）和 PLAIN_MESSAGE（普通）。

2. 输入对话框

JOptionPane 中用于显示输入对话框的方法如下：

- String showInputDialog(Object message)：显示一个 QUESTION_MESSAGE 类型的、指定提示信息的输入对话框。
- String showInputDialog(Component parentComponent,Object message)：在指定父组件上显示一个 QUESTION_MESSAGE 类型的、指定提示信息的输入对话框。
- String showInputDialog(Component parentComponent,Object message,String title,int messageType)：在指定父组件上显示一个指定提示信息、标题和消息类型的输入对话框。

3. 确认对话框

JOptionPane 中用于显示确认对话框的方法如下：

- int showConfirmDialog(Component parentComponent,Object message)：显示一个指定提示信息的、选项类型为 YES_NO_CANCEL_OPTION、标题为“Select an Option”的确认对话框。
- int showConfirmDialog(Component parentComponent,Object message,String title,int optionType)：显示一个指定提示信息、选项类型和标题的确认对话框。

参数 optionType 代表选项类型，它决定对话框中有哪几个按钮选项。

JOptionPane 类中提供了 4 种选项类型：DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION 和 OK_CANCEL_OPTION。

4. 选项对话框

JOptionPane 中用于显示选项对话框的方法如下：

int showOptionDialog(Component parentComponent,Object message,String title,int optionType, int messageType,Icon icon,Object[] options,Object initialValue)：创建一个指定各参数的选项对话框。

下述代码用于实现任务描述 5.D.2，显示各种标准对话框，代码如下：

【描述 5.D.2】OptionPaneDemo.java

```
public class OptionPaneDemo extends JFrame implements ActionListener {
    private JPanel p;
    private JButton btnInput, btnMsg, btnConfirm, btnOption;
    private JTextField txtResult;
    public OptionPaneDemo() {
        super("标准对话框");
        p = new JPanel();
        btnInput = new JButton("输入");
        btnMsg = new JButton("消息");
        btnConfirm = new JButton("确认");
        btnOption = new JButton("选项");
        txtResult = new JTextField(20);
        // 注册监听
        btnInput.addActionListener(this);
        btnMsg.addActionListener(this);
        btnConfirm.addActionListener(this);
        btnOption.addActionListener(this);
        p.add(btnInput);
        p.add(btnMsg);
        p.add(btnConfirm);
        p.add(btnOption);
        p.add(txtResult);
        this.add(p);
        this.setSize(300, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        // 获取事件源
        Object source = e.getSource();
        if (source == btnInput) {
            String strIn = JOptionPane.showInputDialog(btnInput, "请输入姓名：");
            txtResult.setText(strIn);
        }
        if (source == btnMsg) {
            JOptionPane.showMessageDialog(btnMsg, "这是一个消息对话框", "提示",
```

```
                JOptionPane.INFORMATION_MESSAGE);
        }
        if (source == btnConfirm) {
            int r = JOptionPane.showConfirmDialog(btnConfirm, "您确定要删除吗？",
            "删除",JOptionPane.YES_NO_OPTION);
            if (r == JOptionPane.YES_OPTION) {
                txtResult.setText("删除！");
            }
        }
        if (source == btnOption) {
            Object[] options = { "Red", "Green", "Blue" };
            int sel = JOptionPane.showOptionDialog(btnOption, "选择颜色：",
            "选择", JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
             null,options, options[0]);
            if (sel != JOptionPane.CLOSED_OPTION) {
                txtResult.setText("颜色: " + options[sel]);
            }
        }
    }
    public static void main(String[] args) {
        OptionPaneDemo f = new OptionPaneDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 JOptionPane 的 4 个不同的方法显示不同的对话框。因为 4 个按钮注册同一个监听对象，所以事件处理时，先使用 getSource()方法获取事件源，再根据不同的事件源进行不同的处理。

执行结果如图 5-4 所示。



图 5–4　对话框界面

5.3.2 对话框

在 5.3.1 节中，介绍了使用 JOptionPane 类显示标准对话框的方法，本节将讲述手工创建对话框的方法。

JDialog 对话框类可实现一个对话框对象，其常用方法及功能见表 5-6。

表 5–6 JDialog 的常用方法及功能

方法	功能说明
JDialog()	构造方法，用于创建一个无拥有者、无标题、无模式的对话框
JDialog(Frame owner)	构造方法，用于创建一个指定拥有者窗体、无标题、无模式的对话框
JDialog(Frame owner,String title)	构造方法，用于创建一个指定拥有者窗体、指定标题的无模式对话框
JDialog(Frame owner,String title, boolean modal)	构造方法，用于创建一个指定拥有者窗体、标题的模式对话框
add(Componet c)	添加组件到对话框中

实现一个对话框，通常需要继承 JDialog 类，与继承 JFrame 相似，具体过程如下：

01 继承 JDialog 类，在构造方法中传入参数。
02 在对话框中添加用户界面组件。
03 添加事件处理。
04 设置对话框大小。

下述代码用于实现任务描述 5.D.3，使用模式对话框添加新用户，代码如下：

【描述 5.D.3】DialogDemo.java

```
public class DialogDemo extends JFrame implements ActionListener {
    private JMenuBar menuBar;
    private JMenu menuManage;
    private JMenuItem miAddUser;
    // 声明一个滚动面板
    private JScrollPane sp;
    private JTextArea txtContent;
    // 声明对话框
    private UserDialog userDialog;
    public DialogDemo() {
        super("对话框");
        menuBar = new JMenuBar();
        this.setJMenuBar(menuBar);
        menuManage = new JMenu("用户管理");
        menuBar.add(menuManage);
        miAddUser = new JMenuItem("添加新用户");
        menuManage.add(miAddUser);
```

```
        // 注册监听
        miAddUser.addActionListener(this);
        txtContent = new JTextArea(20, 30);
        // 实例化一个滚动面板，置入文本域
        sp = new JScrollPane(txtContent);
        // 将滚动面板添加到窗体中
        this.add(sp);
        this.setSize(300, 200);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 重写事件处理方法
    public void actionPerformed(ActionEvent e) {
        // 判断是否实例化对话框
        if (userDialog == null) {
            userDialog = new UserDialog(this);
        }
        // 显示对话框
        userDialog.setVisible(true);
    }
    public static void main(String[] args) {
        DialogDemo f = new DialogDemo();
        f.setVisible(true);
    }
    // 创建一个对话框类
    class UserDialog extends JDialog implements ActionListener {
        JPanel p;
        JLabel lblName, lblPwd, lblType;
        JTextField txtName;
        JPasswordField txtPwd;
        JComboBox cmbType;
        JButton btnOK, btnCancle;
        public UserDialog(JFrame f) {
            super(f, "添加新用户", true);
            p = new JPanel(new GridLayout(4, 2));
            lblName = new JLabel("用户名");
            lblPwd = new JLabel("密码");
            lblType = new JLabel("类型");
            txtName = new JTextField(10);
            txtPwd = new JPasswordField(10);
            String str[] = { "普通用户", "VIP 用户", "管理员" };
            cmbType = new JComboBox(str);
            btnOK = new JButton("确定");
```

```
            btnCancle = new JButton("取消");
            // 注册监听
            btnOK.addActionListener(this);
            btnCancle.addActionListener(this);
            p.add(lblName);
            p.add(txtName);
            p.add(lblPwd);
            p.add(txtPwd);
            p.add(lblType);
            p.add(cmbType);
            p.add(btnOK);
            p.add(btnCancle);
            this.add(p);
            // 设置合适的大小
            this.pack();
        }
        // 重写事件处理方法
        public void actionPerformed(ActionEvent e) {
            // 单击“确定”按钮时
            if (e.getSource() == btnOK) {
                // 获取用户名文本栏的信息
                String strName = txtName.getText();
                // 获取密码栏中的信息
                String strPwd = new String(txtPwd.getPassword());
                // 验证
                if (strName.equals("")) {
                    JOptionPane.showMessageDialog(btnOK, "用户名不能为空！");
                    return;
                }
                if (strPwd.equals("")) {
                    JOptionPane.showMessageDialog(btnOK, "密码不能为空！");
                    return;
                }
                if (strPwd.length() < 6 || strPwd.length() > 10) {
                JOptionPane.showMessageDialog(btnOK, "密码长度应在 6~10 之间！");
                return;
                }
                // 以追加的方式在文本域中显示信息
                txtContent.append("用户名：" + strName + "\n");
                txtContent.append("密码：" + strPwd + "\n");
                txtContent.append("类型：" + cmbType.getSelectedItem().toString()
                    + "\n\n");
                //隐藏
                this.setVisible(false);
            }
```

```
            // 单击“取消”按钮时
            if (e.getSource() == btnCancle) {
                // 清空文本栏
                txtName.setText("");
                txtPwd.setText("");
                // 隐藏
                this.setVisible(false);
            }
        }
    }
}
```

上述代码中创建了一个继承 JDialog 的内部类，该类的应用和窗体类似，也有组件和事件处理。执行结果如图 5-5 所示。

单击“添加新用户”菜单，弹出如图 5-6 所示的对话框。



图 5–5　主窗口界面



图 5–6　对话框界面

输入用户名、密码，选择类型，单击“确定”按钮，则结果如图 5-7 所示。



图 5–7　添加用户后的结果显示

5.3.3　文件对话框

JFileChooser 类可用于生成打开和保存文件时所显示的对话框，为用户选择文件提供了一种简单的机制，使文件操作变得更加方便。JFileChooser 类的常用方法及功能见表 5-7。

表 5–7　JfileChooser 类的常用方法及功能

方法	功能说明
JFileChooser()	构造方法，用于创建一个默认路径的文件对话框

（续表）

方法	功能说明
JFileChooser(String currentDirectoryPath)	构造方法，用于创建一个指定路径的文件对话框
int showOpenDialog(Component parent)	显示打开文件对话框
int showSaveDialog(Component parent)	显示保存文件对话框
File getSelectedFile()	获取选中的文件对象
File getCurrentDirectory()	获取当前文件路径

下述代码用于实现任务描述 5.D.4，实现记事本功能。

【描述 5.D.4】Note.java

```
public class Note extends JFrame implements ActionListener {
    private JMenuBar menuBar;
    private JMenu menuFile, menuHelp;
    private JMenuItem miNew, miOpen, miSave, miExit, miAbout;
    private JScrollPane sp;
    private JTextArea txtContent;
    public Note() {
        super("记事本");
        menuBar = new JMenuBar();
        this.setJMenuBar(menuBar);
        menuFile = new JMenu("文件");
        menuHelp = new JMenu("帮助");
        menuBar.add(menuFile);
        menuBar.add(menuHelp);
        miNew = new JMenuItem("新建");
        miOpen = new JMenuItem("打开");
        miSave = new JMenuItem("保存");
        miExit = new JMenuItem("退出");
        miAbout = new JMenuItem("关于");
        menuFile.add(miNew);
        menuFile.add(miOpen);
        menuFile.add(miSave);
        menuFile.addSeparator();
        menuFile.add(miExit);
        menuHelp.add(miAbout);
        // 注册监听
        miNew.addActionListener(this);
        miOpen.addActionListener(this);
        miSave.addActionListener(this);
        miExit.addActionListener(this);
        miAbout.addActionListener(this);
```

```
        txtContent = new JTextArea(20, 30);
        sp = new JScrollPane(txtContent);
        this.add(sp);
        this.setSize(400, 300);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == miNew) {
            // 清空文本域中的内容
            txtContent.setText("");
        }
        if (source == miOpen) {
            // 清空文本域中的内容
            txtContent.setText("");
            // 调用打开文件方法
            openFile();
        }
        if (source == miSave) {
            // 调用保存文件方法
            saveFile();
        }
        if (source == miExit) {
            // 系统退出
            System.exit(0);
        }
        if (source == miAbout) {
            JOptionPane.showMessageDialog(this,
                "版本：20100316\n 作者：赵克玲\n 版权所有：海尔软件", "关于",
                JOptionPane.WARNING_MESSAGE);
        }
    }
    // 打开文件的方法
    private void openFile() {
        // 实例化一个文件对话框对象
        JFileChooser fc = new JFileChooser();
        // 显示文件打开对话框
        int rVal = fc.showOpenDialog(this);
        // 如果单击"确定"(Yes/OK)按钮
        if (rVal == JFileChooser.APPROVE_OPTION) {
            // 获取文件对话框中用户选中的文件名
            String fileName = fc.getSelectedFile().getName();
```

```
            // 获取文件对话框中用户选中的文件所在的路径
            String path = fc.getCurrentDirectory().toString();
            try {
                // 创建一个文件输入流，用于读文件
                FileReader fread = new FileReader(path + "/" + fileName);
                // 创建一个缓冲流
                BufferedReader bread = new BufferedReader(fread);
                // 从文件中读一行信息
                String line = bread.readLine();
                // 循环读文件中的内容，并显示到文本域中
                while (line != null) {
                    txtContent.append(line + "\n");
                    // 读下一行
                    line = bread.readLine();
                }
                bread.close();
                fread.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    // 保存文件的方法
    private void saveFile() {
        // 实例化一个文件对话框对象
        JFileChooser fc = new JFileChooser();
        // 显示文件保存对话框
        int rVal = fc.showSaveDialog(this);
        // 如果单击"确定"(Yes/OK)按钮
        if (rVal == JFileChooser.APPROVE_OPTION) {
            // 获取文件对话框中用户选中的文件名
            String fileName = fc.getSelectedFile().getName();
            // 获取文件对话框中用户选中的文件所在的路径
            String path = fc.getCurrentDirectory().toString();
            try {
                // 创建一个文件输出流，用于写文件
                FileWriter fwriter = new FileWriter(path + "/" + fileName);
                // 将文本域中的信息写入文件中
                fwriter.write(txtContent.getText());
                fwriter.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
```

```
        }
    }
    public static void main(String[] args) {
        Note f = new Note();
        f.setVisible(true);
    }
}
```

上述代码实现了记事本的读文件和写文件的功能。在读写文件时，使用文件打开对话框或文件保存对话框进行提示，增强界面的友好性。

执行结果如图 5-8 所示。

单击“文件”→“打开”菜单，弹出“打开”对话框，如图 5-9 所示。

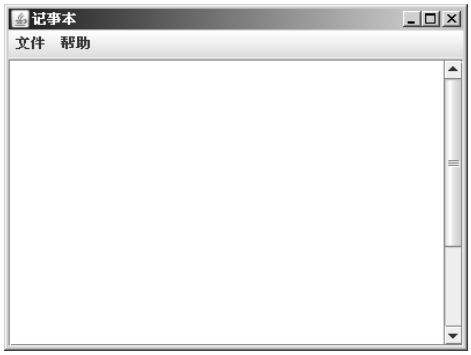


图 5–8　“记事本”窗口



图 5–9　“打开”对话框

单击“文件”→ “保存”菜单，弹出“保存”对话框，如图 5-10 所示。



图 5–10　“保存”对话框

5.3.4　颜色对话框

JColorChooser 是颜色选择器类，提供一个用于允许用户操作和选择颜色的控制器窗格，

在此窗格中可以选取颜色值，其常用的构造方法及功能见表 5-8。

表 5–8　JColorChooser 的构造方法及功能

方法	功能说明
JColorChooser()	创建一个初始颜色为白色的颜色选择器对象
JColorChooser(Color initialColor)	创建一个指定初始颜色的颜色选择器对象
JColorChooser(ColorSelectionModel model)	创建一个指定 ColorSelectionModel 的颜色选择器对象

JColorChooser 是一个组件，而不是对话框，但该类中提供了一个用于创建颜色对话框的静态方法，该对话框包含颜色选择器组件。方法如下所示：

static JDialog createDialog(Component c,String title,boolean modal, JColorChooser chooser, ActionListener okListener,ActionListener cancelListener)

参数说明如下：

- c：对话框的父组件。
- title：对话框的标题。
- modal：是否是模式对话框，值为 true 则是模式对话框形式。
- chooser：颜色选择器对象。
- okListener：对话框中“确定”按钮的事件监听处理对象。
- cancelListener：对话框中“取消”按钮的事件监听处理对象。

下面示例演示 JColorChooser 的使用：

【代码 5–3】ColorChooserDemo.java

```
public class ColorChooserDemo extends JFrame implements ActionListener {
    private JPanel p;
    private JButton b;
    // 声明颜色选取器
    private JColorChooser ch;
    // 声明一个存放颜色的对话框
    private JDialog colorDialog;
    public ColorChooserDemo() {
        super("颜色对话框");
        p = new JPanel();
        b = new JButton("改变面板背景颜色");
        b.addActionListener(this);
        // 实例化颜色选取器对象
        ch = new JColorChooser();
        // 创建一个颜色对话框，颜色选取器对象作为其中的一个参数
        colorDialog = JColorChooser.createDialog(this, "选取颜色", true, ch, null,
                null);
```

```
        p.add(b);
        this.add(p);
        this.setSize(200, 150);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        // 显示颜色对话框
        colorDialog.setVisible(true);
        // 设置面板背景颜色为用户选取的颜色
        p.setBackground(ch.getColor());
    }
    public static void main(String[] args) {
        ColorChooserDemo f = new ColorChooserDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 JColorChooser 创建颜色选择器和颜色对话框对象，用户可以在颜色对话框中选择不同的颜色，来改变面板的背景颜色。通过 getColor()方法可以获取颜色选择器中的颜色值，该方法返回一个 Color 类型的值。

执行结果如图 5-11 所示。

单击“改变面板背景颜色”按钮，弹出“选取颜色”对话框，如图 5-12 所示。



图 5-11　ColorChooserDemo 执行结果

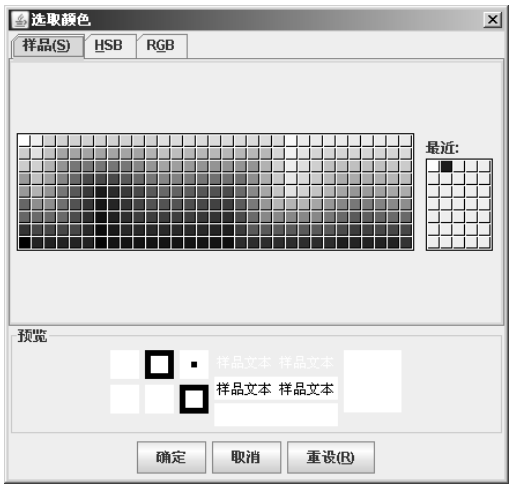


图 5-12　“选取颜色”对话框

5.4 JTable 类

5.4.1 表格

JTable 类用于创建一个表格对象，显示和编辑常规的二维单元表。其构造方法及功能见表 5-9。

表 5-9 Jtable 的构造方法及功能

方法	功能说明
JTable()	创建一个默认模型的表格对象
JTable(int numRows, int numColumns)	创建一个默认模型的、指定行数和列数的表格对象
JTable(Object[][] rowData, Object[]columnNames)	创建一个默认模型的、显示二维数组数据的表格，且可以显示列的名称
JTable(TableModel dm)	创建一个指定表格模型的表格对象
JTable(TableModel dm, TableColumnModel cm)	创建一个指定表格模型、列模型的表格对象
JTable(Vector rowData, Vector columnNames)	创建一个以 Vector 为输入来源的数据表格，显示列名

下面示例演示 JTable 的使用。

【代码 5-4】TableDemo.java

```
public class TableDemo extends JFrame {
    private Object[][] cells = { { "张三", 24, "男", "否" },
            { "李四", 60, "男", "是" }, { "王五", 63, "女", "是" },
            { "刘六", 33, "女", "否" } };
    private String[] columnNames = { "姓名", "年龄", "性别", "婚否" };
    // 声明一个表格对象
    private JTable table;
    private JScrollPane sp;
    public TableDemo() {
        super("表格");
        // 实例化表格对象
        table = new JTable(cells, columnNames);
        // 实例化滚动面板，表格置入其中
        sp = new JScrollPane(table);
        this.add(sp);
        this.setSize(300, 200);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

```
    public static void main(String[] args) {
        TableDemo f = new TableDemo();
        f.setVisible(true);
    }
}
```

在上述代码中，使用 JTable 创建了一个表格，表格需放到滚动面板中，当表格数据内容多时可以滚动，而且列名始终在表格顶部。

执行结果如图 5-13 所示。

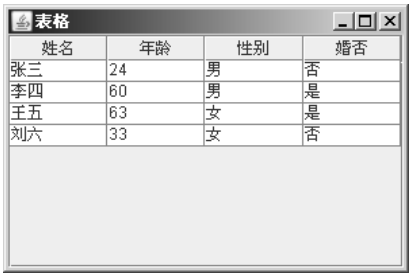


图 5–13　TableDemo 的执行结果

5.4.2　表格模型

TableModel 表格模型是一个接口，此接口定义在 javax.swing.table 包中。在该接口中定义了许多表格操作方法，见表 5-10。

表 5–10　TableModel 接口中的方法

方法	功能说明
void addTableModelListener(TableModelListener l)	注册 TableModelEvent 监听
Class getColumnClass(int columnIndex)	返回列数据类型的类名称
int getColumnCount()	返回列数量
String getColumnName(int columnIndex)	返回指定下标列的名称
int getRowCount()	返回行数
Object getValueAt(int rowIndex,int columnIndex)	返回指定单元格（cell）的值
boolean isCellEditable(int row,int column)	返回单元格是否可编辑
void removeTableModelListener(TableModelListener l)	移除一个监听
void setValueAt(Object aValue,int row,int column)	设置指定单元格的值

TableModel 本身是一个接口，若要直接实现此接口来建立表格是非常复杂的，因此 Java 中还提供了两个类分别实现了 TableModel 接口：

- AbstractTableModel：是一个抽象类，实现了大部分的 TableModel 方法。使用此抽象类可以弹性地构造出自己的表格模式。
- DefaultTableModel：继承 AbstractTableModel 的类，是默认的表格模式类。

5.4.3　表格列模型

TableColumnModel 是一个有关表格列模式的接口，该接口中的方法见表 5-11。

表 5-11　TableColumnModel 接口中的方法及功能

方法	功能说明
void addColumn(TableColumn aColumn)	添加一个新列
TableColumn getColumn(int columnIndex)	获取指定下标的列
int getColumnCount()	获得表格的列数
int getSelectedColumnCount()	获取选中的列数

TableColumnModel 接口通常不需要直接实现，通过调用 JTable 对象中的 getColumnModel()方法可以获取 TableColumnModel 对象，然后再利用此对象对字段进行设置。

5.4.4　表格选择模式

ListSelectionModel 是一个有关表格选择模式的接口。此接口中定义了 3 个有关表格不同选择模式的常量，如表 5-12 所示。

表 5-12　ListSelectionModel 接口中的常量

常量	功能说明
static final int SINGLE_SELECTION	单一选择模式
static final int SINGLE_INTERVAL_SELECTION	连续区间选择模式
static final int MULTIPLE_INTERVAL_SELECTION	多重选择模式

SINGLE_INTERVAL_SELECTION 和 MULTIPLE_INTERVAL_SELECTION 这两种模式在使用时应配合“Shift”键或“Ctrl”键。

ListSelectionModel 接口通常不需要直接实现，通过调用 JTable 对象的 getSelectionModel()方法可以获取 ListSelectionModel 对象，然后再利用此对象的 setSelectionModel()方法对选择模式进行设置。

当用户对表格进行数据域位的选取时会产生 ListSelectionEvent 事件，要处理此事件就必须实现 ListSelectionListener 监听接口，该接口中定义了一个事件处理方法，如下：

void valueChanged(ListSelectionEvent e)：选取的单元格数据改变时，会自动调用此方法进行处理。

下述代码用于实现任务描述 5.D.5，改变表格选择模式，代码如下：

【描述 5.D.5】SelectionModelDemo.java

```
public class SelectionModelDemo extends JFrame implements ActionListener,
ListSelectionListener{
    private JPanel p;
    private JButton btnSin,btnSinIn,btnMul;
    private JTable table;
    //声明表格选中模式对象
```

```
private ListSelectionModel selectionMode;
private JScrollPane sp;
private JLabel lblMsg;
public SelectionModelDemo() {
    super("表格选择模式");
    p = new JPanel();
    btnSin= new JButton("单一选择");
    btnSinIn = new JButton("连续区间选择");
    btnMul = new JButton("多重选择");
    btnSin.addActionListener(this);
    btnSinIn.addActionListener(this);
    btnMul.addActionListener(this);
    p.add(btnSin);
    p.add(btnSinIn);
    p.add(btnMul);
    this.add(p,BorderLayout.NORTH);
    String[] name = { "字段 1", "字段 2", "字段 3", "字段 4", "字段 5" };
    String[][] data = new String[5][5];
    int value = 1;
    for (int i = 0; i < data.length; i++) {
        for (int j = 0; j < data.length; j++) {
            data[i][j] = String.valueOf(value++);
        }
    }
    table = new JTable(data, name);
    table.setPreferredScrollableViewportSize(new Dimension(400, 80));
    //设置表格的选取是以 cell 为单位,而不是以列为单位
    table.setCellSelectionEnabled(true);
    // 获取表格的模式对象
    selectionMode = table.getSelectionModel();
    //注册选取事件监听
    selectionMode.addListSelectionListener(this);
    sp = new JScrollPane(table);
    this.add(sp);
    lblMsg = new JLabel("你选取了:");
    this.add(lblMsg,BorderLayout.SOUTH);
    this.setSize(400, 300);
    this.setLocation(100, 100);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
/**
```

```
     * 处理按钮事件，利用 ListSelectionModel 界面所定义的
     * setSelectionMode()方法来设置表格选取模式
     */
    public void actionPerformed(ActionEvent e) {
        Object source=e.getSource();
        if (source==btnSin) {
            //设置表格的选取模式为单一选择模式
    selectionMode.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        }
        if (source==btnSinIn) {
            //设置表格的选取模式为区域选择模式
    selectionMode.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTI
ON);
        }
        if (source==btnMul) {
            //设置表格的选取模式为多重选择模式
    selectionMode.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELEC
TION);
        }
        //表格重新生效
        table.revalidate();
    }
    /**
     * 当用户选取表格数据时会触发 ListSelectionEvent
     * 实现 ListSelectionListener 接口来处理这一事件
     * ListSelectionListener 界面只定义一个方法
     * 那就是 valueChanged()
     */
    public void valueChanged(ListSelectionEvent el) {
        String tempString = "";
        // 获取所有选中行的下标
        int[] rows = table.getSelectedRows();
        //获取所有选中列的下标
        int[] columns = table.getSelectedColumns();
        //循环取出选中的数据，JTable 的 getValueAt()方法会返回指定行列的 cell 数据
        for (int i = 0; i < rows.length; i++) {
            for (int j = 0; j < columns.length; j++) {
                tempString = tempString + " "+ (String) table.getValueAt(rows[i],
columns[j]);
            }
            lblMsg.setText("你选取:" + tempString);
```

```
        }
    }
    public static void main(String[] args) {
        SelectionModelDemo f = new SelectionModelDemo();
        f.setVisible(true);
    }
}
```

上述代码实现了表格的不同选择模式，并将选择的所有单元格信息显示在标签中。

执行结果如图 5-14 所示。



图 5–14　SelectionModelDemo 的执行结果

5.5　JTree 类

5.5.1　树

JTree 类用来创建树目录组件，是一个将分层数据集显示为轮廓的组件。树中的节点可以展开，也可以折叠。当展开一个非叶节点时，将显示其子节点；折叠节点则将其子节点隐藏。

JTree 常用的方法及功能见表 5-13。

表 5–13　JTree 的方法及功能

常量	功能说明
JTree()	创建一个默认模型的 Swing 树对象
JTree(Object[] value)	创建一棵不显示根节点的树，它将指定数组中的每个元素作为根节点的子节点
JTree(Vector value)	创建一棵不显示根节点的树，它将指定向量中的每个元素作为根节点的子节点
TreePath getSelectionPath()	返回首选节点的路径

（续表）

常量	功能说明
void setModel(TreeModel mdl)	设置将提供数据的 TreeModel
void updateUI()	更新 UI

5.5.2 树模型

TreeModel 树的模型接口可以触发相关的树事件，处理树可能产生的一些变动。该接口中定义了如表 5-14 所示的方法。

表 5–14 TreeModel 接口中的方法及功能

方法	功能说明
void addTreeModelListener(TreeModelListener l)	注册树监听
Object getChild(Object parent, int index)	返回子节点
int getChildCount(Object parent)	返回子节点数量
int getIndexOfChild(Object parent, Object child)	返回子节点的索引值
Object getRoot()	返回根节点
boolean isLeaf(Object node)	判断是否为树叶节点
void removeTreeModelListener(TreeModelListener l)	删除 TreeModelListener
void valueForPathChanged(TreePath path, Object newValue)	改变 Tree 上指定节点的值

通过实现 TreeModel 接口中的这 8 种方法，可以构造出自己想要的 JTree，但这种做法比较复杂。Java 中提供了一个 DefaultTreeModel 默认模式类，该类实现了 TreeModel 接口，并提供了许多实用的方法，能方便、快捷地构造出 JTree。DefaultTreeModel 的构造方法及功能见表 5-15。

表 5–15 DefaultTreeModel 的构造方法

方法	功能说明
DefaultTreeModel(TreeNode root)	创建一个 DefaultTreeModel 对象，并指定根节点
DefaultTreeModel(TreeNode root, Boolean asksAllowsChildren)	创建一个指定根节点的、是否具有子节点的 DefaultTreeModel 对象

5.5.3 树节点

TreeNode 是树节点，它是一个接口，里面定义了有关节点的方法，如表 5-16 所示。

表 5–16 TreeNode 接口中常用方法及功能

方法	功能说明
Enumeration children()	获取子节点
TreeNode getChildAt(int childIndex)	返回指定下标的子节点对象
int getChildCount()	返回子节点数量
TreeNode getParent()	返回父节点对象

（续表）

方法	功能说明
int getIndex(TreeNode node)	返回指定节点的下标
boolean getAllowsChildren()	获取是否有子节点
boolean isLeaf()	获取是否为叶节点（无子节点的节点）

DefaultMutableTreeNode 是一个实现 TreeNode 和 MutableTreeNode 接口的类。该类中提供了许多实用的方法，并增加了一些有关节点的处理方式。其常用的方法及功能见表 5-17。

表 5–17　DefaultMutableTreeNode 的常用方法及功能

方法	功能说明
DefaultMutableTreeNode()	构造方法，用于创建一个空的树节点对象
DefaultMutableTreeNode(Object userObject)	构造方法，用于建立一个指定内容对象的树节点
DefaultMutableTreeNode(Object userObject, Boolean allowsChildren)	构造方法，用于建立一个指定内容对象的、是否有子节点的树节点
void add(MutableTreeNode newChild)	添加一个树节点
void insert(MutableTreeNode newChild, int childIndex)	插入一个树节点
void remove(MutableTreeNode aChild)	删除一个树节点
void setUserObject(Object userObject)	设置树节点的内容对象



5.5.4　树事件

树事件有两种类型，分别是 TreeModelEvent 事件和 TreeSelectionEvent 事件。

1. TreeModelEvent 事件

当树的结构改变时，例如：改变节点值、新增节点、删除节点等，都会触发 TreeModelEvent 事件。处理 TreeModelEvent 事件的监听接口是 TreeModelListener，该接口中定义的事件处理方法见表 5-18。

表 5–18　TreeModelListener 接口中定义的事件处理方法及功能

方法	功能说明
void treeNodesChanged(TreeModelEvent e)	节点改变时，调用此事件处理方法
void treeNodesInserted(TreeModelEvent e)	插入节点时，调用此事件处理方法
void treeNodesRemoved(TreeModeEvent e)	删除节点时，调用此事件处理方法
void treeStructureChanged(TreeModelEvent e)	树结构改变时，调用此事件处理方法

2. TreeSelectionEvent 事件

当在 JTree 上选择任何一个节点时，都会触发 TreeSelectionEvent 事件。处理此事件的监听接口是 TreeSelectionListener，该接口中定义的事件处理方法见表 5-19。

表 5-19　TreeSelectionListener 接口中的事件处理方法及功能

方法	功能说明
void valueChanged(TreeSelectionEvent e)	当选择的节点改变时，自动调用此方法进行事件处理

下述代码用于实现任务描述 5.D.6，处理 TreeSelectionEvent 事件。

【描述 5.D.6】TreeDemo.java

```
public class TreeDemo extends JFrame implements TreeSelectionListener {
    private DefaultMutableTreeNode root;
    private DefaultTreeModel model;
    private JTree tree;
    private JTextArea textArea;
    private JPanel p;
    public TreeDemo() {
        super("TreeBrowserTest");
        // 实例化树的根节点
        root = makeSampleTree();
        // 实例化的树模型
        model = new DefaultTreeModel(root);
        // 实例化一棵树
        tree = new JTree(model);
        // 注册树的监听对象，监听选择不同的树节点
        tree.addTreeSelectionListener(this);
        // 设置树的选择模式是单一节点的选择模式（一次只能选中一个节点）
        tree.getSelectionModel().setSelectionMode(
                TreeSelectionModel.SINGLE_TREE_SELECTION);
        // 实例化一个面板对象，布局是 1 行 2 列
        p = new JPanel(new GridLayout(1, 2));
        // 在面板的左侧放置树
        p.add(new JScrollPane(tree));
        textArea = new JTextArea();
        // 在面板的右侧放置文本域
        p.add(new JScrollPane(textArea));
        this.add(p);
        this.setSize(400, 300);
        this.setLocation(100, 100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 创建一棵树对象的方法
    public DefaultMutableTreeNode makeSampleTree() {
        // 实例化树节点，并将节点添加到相应节点中
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("世界");
        DefaultMutableTreeNode country = new DefaultMutableTreeNode("中国");
        root.add(country);
```

```
        DefaultMutableTreeNode state = new DefaultMutableTreeNode("山东");
        country.add(state);
        DefaultMutableTreeNode city = new DefaultMutableTreeNode("青岛");
        state.add(city);
        city = new DefaultMutableTreeNode("济南");
        state.add(city);
        state = new DefaultMutableTreeNode("江苏");
        country.add(state);
        city = new DefaultMutableTreeNode("南京");
        state.add(city);
        return root;
    }
    // 重写树的选择事件处理方法
    public void valueChanged(TreeSelectionEvent event) {
        // 获取选中节点的路径
        TreePath path = tree.getSelectionPath();
        if (path == null)
            return;
        // 获取选中的节点对象
        DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) path
                .getLastPathComponent();
        // 获取选中节点的内容，并显示到文本域中
        textArea.setText(selectedNode.getUserObject().toString());
    }
    public static void main(String[] args) {
        TreeDemo f = new TreeDemo();
        f.setVisible(true);
    }
}
```

上述代码实现了一个窗口界面，左边显示树目录，右边是文本域，当选择不同的树节点时，在文本域中显示节点的文本内容。为了处理 TreeSelectionEvent 事件，继承 TreeModelListener 接口，重写 valueChanged()方法。

执行结果如图 5-15 所示。

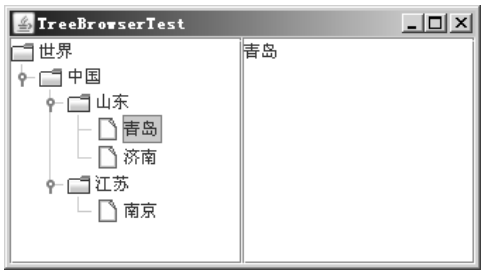


图 5–15　TreeDemo 的执行结果

小结

通过本章的学习，学生应该能够学会：

- 菜单组件之间的层次关系是：菜单选项添加到菜单中，菜单添加到菜单栏（或上级菜单）中，菜单栏添加到窗体容器中。
- JPopupMenu 是一种不固定在菜单栏中、随处浮动的弹出式菜单。
- 标准对话框有：消息对话框、确认对话框、输入对话框及选项对话框 4 种。
- 通过继承 JDialog 类可以实现一个对话框。
- JFileChooser 类可用于实现打开和保存文件时显示的对话框。
- JColorChooser 类可用于创建选择颜色的对话框。
- 用 JTable 创建表格时需要对表格模型进行修饰，表格模型可以使用默认模型或自定义模型。
- JTree 用于实现树形结构，通过设置其树节点事件完成相应的操作。

练习

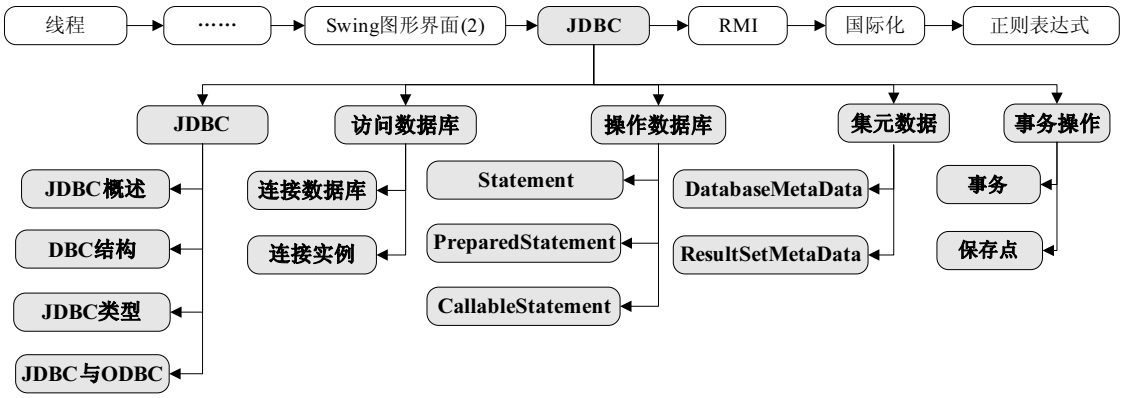
1. 用于创建菜单项的类是_____。
 A. JMenuItem　　B. JPopupMenu　　C. JMenu　　D. JMenuBar
2. 下面选项中，用于显示确认对话框的方法是_____。
 A. showMessageDialog()　　B. showConfirmDialog()
 C. showInputDialog()　　D. showOptionDialog()
3. 下面_____组件用于以行或列的形式显示数据。
 A. JTree　　B. JScrollPane　　C. JTable　　D. JFrame
4. JTable 的父类是_____。
 A. JComponent　　B. JContainer　　C. Component　　D. Container
5. 下面_____组件用于以层次结构显示数据。
 A. JTree　　B. JScrollPane　　C. JTable　　D. JFrame
6. 用于文件打开或保存时显示的对话框类是_____，该类中的_____方法用于显示一个文件打开对话框，_____方法用于显示一个文件保存对话框。
7. _____类是颜色选择器。
8. 简述实现一个自定义对话框的步骤。

第 6 章　JDBC

本章目标

- 理解 JDBC 访问数据库的结构及原理
- 了解 JDBC 四种驱动类型
- 掌握 JDBC 访问数据库的步骤
- 掌握 JDBC 中的 DriverManager 类和 Connection、ResultSet 接口
- 掌握 JDBC 的常用查询接口
- 掌握 JDBC 对集元数据的访问
- 掌握 JDBC 的事务处理

学习导航



任务描述

【描述 6.D.1】

通过连接 Oracle，演示访问数据库的一般步骤。

【描述 6.D.2】

使用 Statement 接口，实现对数据表的 CRUD 操作。

【描述 6.D.3】

使用 PreparedStatement 接口，实现对数据表的动态 CRUD 操作。

【描述 6.D.4】

使用 CallableStatement 实现各种情形存储过程的调用。

【描述 6.D.5】

使用 DatabaseMetaData 获取当前数据库连接的相关信息。

【描述 6.D.6】

使用 ResultSetMetaData 获取当前结果集的相关信息。

【描述 6.D.7】

通过更新操作演示在 JDBC 中的事务控制。

【描述 6.D.8】

在 JDBC 中通过设置保存点，控制事务的部分回滚。

6.1　JDBC 基础

通常数据应用系统的开发都是基于特定的数据库管理系统的，如 Oracle、DB2、MS SQL Server、Sybase、MySQL 等。JDBC 为开发人员提供了新的数据库开发工具。

6.1.1　JDBC 概述

Java 数据库连接（Java Database Connectivity，JDBC），是一种用于执行 SQL 语句的 Java API，它由一组用 Java 编程语言编写的类和接口组成。JDBC 为数据库开发人员提供了一个标准的 API，使他们能够用纯 Java API 来编写数据库应用程序。数据库开发人员使用 JDBC API 编写一个程序后，就可以很方便地将 SQL 语句传送给几乎任何一种数据库，如 Sybase、Oracle 或 MS SQL Server 等。使用 JDBC 编写的程序能够自动地将 SQL 语句传送给相应的数据库管理系统。JDBC 扩展了 Java 的功能，由于 Java 语言本身的特点，使得 JDBC 具有简单、健壮、安全、可移植、获取方便等优势。

6.1.2　JDBC 结构

JDBC API 是 Java 开发工具包（JDK）的组成部分，由 3 个部分组成：

- JDBC 驱动程序管理器
- JDBC 驱动程序测试工具包
- JDBC-ODBC 桥

JDBC 驱动程序管理器是 JDBC 体系结构的支柱，其主要作用是把 Java 应用程序连接到正确的 JDBC 驱动程序上。

JDBC 驱动程序测试工具包为 JDBC 驱动程序的运行提供一定的可信度，只有通过 JDBC 驱动程序测试包的驱动程序才被认为是符合 JDBC 标准的。

JDBC-ODBC 桥使 ODBC 驱动程序可被用做 JDBC 驱动程序。其目标是为方便实现访问某些不常见的 DBMS，它的实现为 JDBC 的快速发展提供了一条途径。

JDBC API 既支持数据库访问的两层模型，同时也支持三层模型。

在两层模型中，Java 应用程序将直接与数据库进行对话。在这种情况下，需要一个 JDBC 驱动程序来与所访问的特定数据库管理系统进行通信。用户的 SQL 语句被直接送往数据库，而处理的结果将被送回给用户。存放数据的数据库可以位于另一台物理计算机上，用户通过网络连接到数据库服务器，这就是典型的客户机/服务器模型（C/S）。两层应用模型如图 6-1 所示。

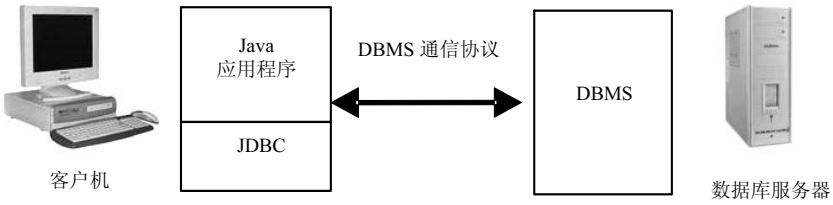


图 6–1　两层应用模型

在三层模型中，客户通过浏览器或 Java 小应用程序、而应用程序通过 JDBC API 提出 SQL 请求，该请求先是被发送到服务的“中间层”，然后由中间层将 SQL 语句转发给数据库，数据库对 SQL 语句进行处理并将结果送回到中间层，中间层再将结果送回给用户。三层应用模型如图 6-2 所示。

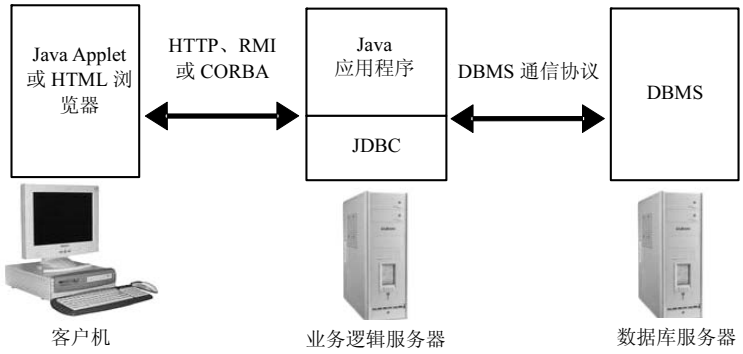


图 6–2　三层应用模型

使用中间层的几个好处是：用户可以利用易于使用的高级 API，由中间层把它转换为相应的低级调用，而不用关心低级调用的复杂细节问题。一定程度上三层模型结构还能提供一些性能上的好处。

6.1.3　JDBC 类型

JDBC 驱动类型可以归结为以下几类：

- JDBC-ODBC 桥加 ODBC 驱动程序
- 本地 API 部分用 Java 来编写的驱动程序
- JDBC 网络纯 Java 驱动程序
- 本地协议纯 Java 驱动程序

JDBC-ODBC 桥接 ODBC 驱动程序将 JDBC 翻译成 ODBC，然后使用一个 ODBC 驱动程序与数据库进行通信。Sun 公司发布的 JDK 中包含一个这样的驱动：JDBC/ODBC 桥。有一点需要注意的是：在使用桥接之前必须将 ODBC 加载到使用该驱动程序的每个客户机上，并对 ODBC 进行相应的部署和设置。因此，这种类型的驱动程序最适合于企业网，或者是用 Java

编写的三层结构的应用程序服务器代码。

本地 API 部分用 Java 来编写的驱动程序把客户机 API 上的 JDBC 调用转换为 Oracle、Sybase、Informix、DB2 或其他 DBMS 的调用，像桥驱动程序一样，这种类型的驱动程序要求将某些二进制代码加载到每台客户机。

JDBC 网络纯 Java 驱动程序将 JDBC 转换为与 DBMS 无关的网络协议，之后这种协议又被某个服务器转换为一种 DBMS 协议。这种网络服务器中间件能够将它的纯 Java 客户机连接到多种不同的数据库上。所用的具体协议取决于提供者，这是最为灵活的 JDBC 驱动程序。

本地协议纯 Java 驱动程序将 JDBC 调用直接转换为 DBMS 所使用的专用网络协议。这将允许从客户机机器上直接调用 DBMS 服务器，是 Intranet 访问的一个很实用的解决方法。

大部分数据库供应商都为他们的产品提供第三类或第四类驱动程序。与数据库供应商提供的程序相比，许多第三方公司开发了更符合标准的产品，它们支持更多的平台、性能更高，一定程度上可靠性也更高。

注意　*实际开发中第三和第四类驱动程序是 JDBC 访问数据库的首选方法，第一和第二类驱动程序可以作为过渡方案来使用。*

6.1.4　JDBC 与 ODBC

在数据库访问的发展中，ODBC（开放式数据库连接）API 可能是使用最广泛的用于访问关系数据库的编程接口，它几乎可以将所有平台的关系型数据库连接起来，其模型结构如图 6-3 所示。

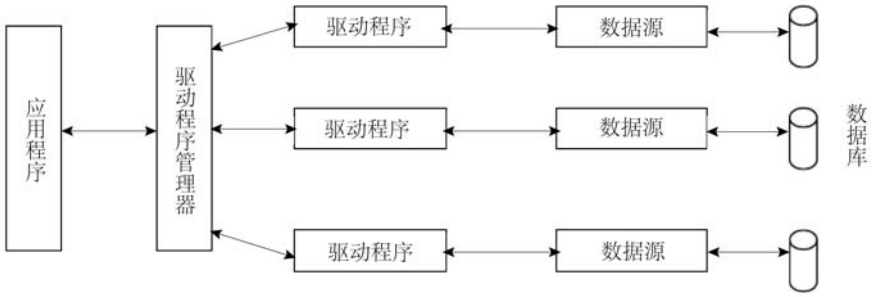


图 6–3　ODBC 模型结构

在 Java 中可以在 JDBC 的帮助下通过 JDBC-ODBC 桥接方式实现 ODBC 功能调用，其实现方式如图 6-4 所示，之所以不直接使用 ODBC 主要是基于以下几个原因：

- ODBC 是用 C 语言实现的，如果在 Java 中直接调用 C 语言代码将带来健壮性、安全性、移植性等方面的问题。
- 由于在 ODBC 中使用了大量容易出错和造成安全问题的游标，而 Java 取消了游标，因此无法将 ODBC API 全部翻译成 JDBC API。
- ODBC 将简单和高级功能混在一起，即使简单的查询也要使用复杂的选项。JDBC 在

设计上尽量做到简单化，提倡简单的事用简单的方法实现，高级功能只有在必要时才会使用，因此 JDBC 比 ODBC 更容易学习。

- 使用 ODBC 时，ODBC 驱动程序管理器与驱动程序都必须手工装入到每台客户机系统中。而使用 JDBC 时，因为 JDBC 驱动程序全部是由 Java 编写的代码，所以在所有 Java 平台本地机或网络客户机上都可以自动安装。



图 6-4　JDBC-ODBC 桥接模型

总之，JDBC API 是基于 SQL 抽象和概念设计的一种自然的纯 Java 的解决方案。JDBC 建立在 ODBC 上，保留了 ODBC 的基本设计特征，以 Java 风格与优点为基础并进行优化，因此更加易于使用。

注意　伴随 JDK 版本的变更，目前最新的 JDBC 版本为 JDBC4.0，本章以 JDBC3.0 规范为依据进行介绍。

6.2　访问数据库

本章使用 Oracle 数据库作为 JDBC 的访问环境，需做如下准备工作：

- 设置 Oracle 驱动的类路径。在使用 Oracle 数据库的时候，需要使用它所提供的驱动，在安装完 Oracle 数据库后，可以在 Oracle 的安装路径下找到，对于 Oracle10g（Oracle9i 也类似），可以在<oracle_home>\product\10.1.0\Db_2\jdbc\lib 目录下，找到 ojdbc14.jar 包，将此工具包导入到工程中，或加入到环境变量 CLASSPATH 中即可。
- 创建演示表格及测试数据。

讲解数据库操作，需要创建数据表及演示数据。在 Scott 方案中创建 Student 表，并添加测试数据的代码如下：

```
create table student(
sno varchar2(5) primary key,--学生号
name varchar2(10) not null,--学生姓名
age  number(2) not null,--学生年龄
sex  char(1) not null --性别,1：男,0：女
);
```

```
--添加测试数据
insert into student values('95001','张飞',23,1);
insert into student values('95002','关羽',23,1);
insert into student values('95003','貂蝉',23,0);
insert into student values('95005','小乔',23,0);
```

6.2.1　连接数据库

使用 JDBC 访问数据库的基本步骤一般如下：

01 加载 JDBC 驱动程序。
02 建立数据库连接。
03 创建 Statement 对象。
04 执行 SQL 语句。
05 处理返回结果。
06 关闭创建的对象。

1. 加载 JDBC 驱动程序

在与某一特定数据库建立连接前，首先应加载一种可用的 JDBC 驱动程序。JDBC 可用的驱动程序有 JDBC-ODBC 桥接驱动、JDBC 网络驱动器或是由特定数据库厂商提供的驱动程序等。加载驱动程序的一种简单方法是使用 Class.forName 方法显式加载：

```
Class.forName("DriverName");
```

其中 DriverName 是要加载的 JDBC 驱动程序名称。驱动程序名称根据数据库厂商提供的 JDBC 驱动程序的种类来确定。

对于 JDBC/ODBC 桥，加载 JDBC-ODBC 数据库驱动程序的方法为：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

而对于 Oracle 数据库，加载数据库驱动程序的方法为：

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. 创建数据库连接

DriverManager 类是 JDBC 的管理层，作用于用户和驱动程序之间。它跟踪可用的驱动程序，并在数据库和相应驱动程序之间建立连接。该类负责加载、注册 JDBC 驱动程序，管理应用程序和已注册的驱动程序的连接。其常用方法及使用说明见表 6-1。

表 6–1　DriverManager 类的方法列表

方法名	功能说明
Static connection getConnection(String url, String user, String password)	用于建立到指定数据库 URL 的连接。其中 url 提供了一种标识数据库位置的方法，user 为用户名，password 为密码
static Driver getDriver(String url)	用于返回能够打开 url 所指定的数据库的驱动程序

使用 JDBC 操作数据库之前，首先必须创建一个数据库连接，此时需要使用 DriverManager 类的 getConnection 方法，其一般的使用格式如下：

```
Connection conn = DriverManager.getConnection(String url,String user,String password);
```

这里的 url 提供了一种标识数据库位置的方法，可以使相应的驱动程序能够识别该数据库并与它建立连接。JDBC URL 由三个部分组成，各个部分之间用冒号分隔，格式如下：

```
jdbc : < subprotocol> : < subname>
```

其中，<subprotocol>是子协议，指数据库连接的方式。<subname>可以根据子协议的改变而变化。对于 JDBC-ODBC 桥驱动的连接 URL 是：

```
jdbc:odbc:<data_source>[<attribute_name1>=<attribute_value1>] ……
```

其中，odbc 为子协议名称，data_source 指本地 ODBC 数据源的名字，attribute_name 和 attribute_value 用于指定建立连接所必需的信息，如用户名、密码等。

通过 ODBC 获取数据库连接的示例语句如下：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection Conn=
    DriverManager.getConnection("jdbc:odbc:myscott","scott","tiger");
```

对于 Oracle 驱动连接的 URL 是：

```
jdbc:oracle:thin:@serverName:port:instance
```

其中，oracle 为子协议名称，thin 是 oracle 数据库的一种连接方式，serverName 为 Oracle 数据库服务器名称，可以是一个域名，也可以是 IP 地址，port 为 Oracle 数据库的端口号，默认为 1521，instance 是数据库的实例名。

通过 Oracle 驱动获取数据库连接的示例语句如下：

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection Conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
```

DriverManager 类的 getConnection()方法返回一个 Connection 对象。Connection 是一个接口，表示与数据库的连接，并拥有创建 SQL 语句的方法，以完成基本的 SQL 操作，同时为数据库事务处理提供提交和回滚的方法。一个应用程序可与单个数据库建立一个或多个连接，也可以与多个数据库建立连接。Connection 接口的常用方法及使用说明见表 6-2。

表 6-2　Connection 接口的方法列表

方法名	功能说明
void close()	断开连接，释放此 Connection 对象的数据库和 JDBC 资源
Statement createStatement()	创建一个 Statement 对象来将 SQL 语句发送到数据库
void commit()	用于提交 SQL 语句，确认从上一次提交/回滚以来进行的所有更改
boolean isClosed()	用于判断 Connection 对象是否已经被关闭
CallableStatement prepareCall(String sql)	创建一个 CallableStatement 对象来调用数据库存储过程
PreparedStatement prepareStatement(String sql)	创建一个 PreparedStatement 对象来将参数化的 SQL 语句发送到数据库
void rollback()	用于取消 SQL 语句，取消在当前事务中进行的所有更改

注意　为了提供更好的可移植性，可以将相关的连接数据保存到属性（Properties）文件中，例如，我们可以将连接 Oracle 操作的相关配置放到一个 oracle.properties 属性文件中去，例如：

```
driver = oracle.jdbc.driver.OracleDriver
url = jdbc:oracle:thin:@localhost:1521:orcl
user = scott
pwd = tiger
```

然后在程序中将这些信息读取出来，作为连接数据库的相关参数。

3. 创建 Statement

创建完连接之后，可以通过此连接向目标数据库发送 SQL 语句。在发送 SQL 语句之前，必须创建一个 Statement 类的对象，该对象负责将 SQL 语句发送给数据库。如果 SQL 语句运行后产生结果集，Statement 对象会将结果集封装成 ResultSet 对象并返回。

通过调用 Connection 接口的 createStatement 方法来创建 Statement 对象，如：

```
Statement smt = conn.createStatement();
```

4. 执行 SQL 语句

获取 Statement 对象之后，就可以使用该对象的 executeQuery()方法来执行 SQL 语句，如：

```
ResultSet rs = smt.executeQuery("SELECT sno,name,age FROM student");
```

executeQuery()方法返回一个 ResultSet 对象，它包含了 SQL 的查询结果。

5. 处理返回结果

在 JDBC 中，SQL 的查询结果使用 ResultSet 封装。通过 Statement 对象的 executeQuery()方法封装并返回。ResultSet 对象维持了执行某个 SQL 语句后满足条件的所有的行，它还提供了一系列方法完成对结果集中数据的操作。

6. 关闭创建的对象

当数据库操作执行完毕或退出应用前，需将数据库访问过程中建立的对象按顺序关闭，防止系统资源浪费。关闭的次序是：

01 关闭结果集。
02 关闭 Statement 对象。
03 关闭连接。

6.2.2 连接实例

分析访问数据库的步骤，可以发现，除创建 Statement、执行 SQL 语句和处理返回结果的过程随应用不同外，其余的操作几乎是相同的。为方便后期使用，可以抽取其共同特性，将连接的创建和对象关闭操作通过静态方法实现，创建如下工具类：

【代码 6-1】 DBUtil.java

```
public class DBUtil {
    // 获取数据库连接
    public static Connection getConnection() throws SQLException {
        try {
            // 指定驱动程序
            Class.forName(Constants.DRIVER);
            // 建立数据库连接
            Connection conn = DriverManager.getConnection(Constants.URL,
                    Constants.USERNAME, Constants.PASSWORD);
            return conn;
        } catch (Exception e) {
            // 如果连接过程出现异常，抛出异常信息
            throw new SQLException("驱动错误或连接失败！");
        }
    }
    //关闭 Statement 对象
    public static void closeStatement(Statement stmt) throws SQLException {
        try {
            // 关闭 stmt
            if (stmt != null)
                stmt.close();
        } catch (SQLException e) {
            throw new SQLException("关闭 Statement 失败！");
        }
    }
    //关闭 ResultSet 对象
    public static void closeResultSet(ResultSet rs) throws SQLException {
```

```
        try {
            // 关闭 rs
            if (rs != null)
                rs.close();
        } catch (SQLException e) {
            throw new SQLException("关闭 ResultSet 失败！");
        }
    }
    //关闭 Connection 对象
    public static void closeConnection(Connection conn) throws SQLException {
        try {
            // 关闭 rs
            if (conn != null)
                conn.close();
        } catch (SQLException e) {
            throw new SQLException("关闭 Connection 失败！");
        }
    }
}
```

上述代码中，在创建数据库连接时使用了静态常量定义相关的数据库连接信息，代码如下：

【代码 6–2】 Constants.java

```
public class Constants {
    //Oracle 数据库连接信息
    public static String DRIVER = "oracle.jdbc.driver.OracleDriver";
    public static String URL = "jdbc:oracle:thin:@localhost:1521:orcl";
    public static String USERNAME = "scott";
    public static String PASSWORD = "tiger";
}
```

使用静态常量的好处是便于维护和修改。

注意　本章所介绍内容和演示实例，都将通过 Oracle 驱动进行 JDBC 操作。

下述代码用于实现任务描述 6.D.1，通过连接 Oracle，演示访问数据库的一般步骤。

【描述 6.D.1】 ConnectionDemo.java

```
public class ConnectionDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        try {
            // 通过工具类获取数据库连接
```

```
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            // 判断数据库是否处于打开状态
            // System.out.println(conn.isClosed());
            // 创建 Statment 对象
            Statement stmt = conn.createStatement();
            // 获取查询结果集
            ResultSet rs = stmt
                    .executeQuery("SELECT sno,name,age FROM student");
            System.out.println("查询成功！");
            while (rs.next()) {
                System.out.println(rs.getString(1));
            }
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
连接成功!
查询成功!
95001
95002
95003
95005
```

注意 在使用 JDBC 进行数据库查询操作时，不建议使用“select * from student”方式的 SQL 语句，应在查询语句中指明具体列。

6.3 操作数据库

在JDBC中要执行SQL查询语句可以通过执行一般查询、参数查询和存储过程三种方式。这三种方式分别对应 Statement、PreparedStatement 和 CallableStatement 接口。

6.3.1 Statement 接口

Statement 接口一般用于执行静态的 SQL 语句。静态的 SQL 语句在执行时不需要接收任何参数，其主要功能是将 SQL 语句传送给数据库，并将 SQL 语句的执行结果返回。

提交的 SQL 语句可以是：

- SELECT 查询语句
- SQL DML 语句，如 INSERT、UPDATE 或 DELETE。
- SQL DDL 语句，如 CREATE TABLE 和 DROP TABLE。

Statement 接口的常用方法及功能见表 6-3。

表 6-3　Statement 接口的常用方法列表

方法名	功能说明
void close()	关闭 Statement 对象
boolean execute(String sql)	执行给定的 SQL 语句，该语句可能返回多个结果
ResultSet executeQuery(String sql)	执行给定的 SQL 语句，该语句返回单个 ResultSet 对象
int executeUpdate(String sql)	执行给定 SQL 语句，该语句可能为 INSERT、UPDATE 或 DELETE 语句，或者不返回任何内容的 SQL 语句（如 SQL DDL 语句）
Connection getConnection()	获取生成此 Statement 对象的 Connection 对象
int getFetchSize()	获取结果集合的行数，该数是根据此 Statement 对象生成的 ResultSet 对象的默认获取大小
int getMaxRows()	获取由此 Statement 对象生成的 ResultSet 对象可以包含的最大行数

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery()、executeUpdate()和 execute()。使用哪一个方法由 SQL 语句所产生的内容决定：

- executeQuery()方法用于产生单个结果集的语句，例如 SELECT 语句。
- executeUpdate()方法用于执行 INSERT、UPDATE 或 DELETE 语句及 SQL DDL 语句，如 CREATE TABLE 和 DROP TABLE。INSERT、UPDATE 或 DELETE 语句的效果是修改表中的行或列，executeUpdate 的返回值是一个整数，指示受影响的行数（即更新计数）。对于 CREATE TABLE 或 DROP TABLE 等语句，executeUpdate 的返回值总为零。
- execute()方法用于执行返回多个结果集、多个更新计数或二者的组合。此方法比较特殊，一般是在用户不知道执行 SQL 语句后会产生什么结果或可能有多种类型的结果产生时才会使用。

ResultSet 封装了 Statement 的 executeQuery()方法返回的结果集，即符合 SQL 语句中指定条件的所有行。ResultSet 维护了指向当前行的游标，并提供了许多方法来操作结果集中的游标，同时提供了一套 getXXX()方法对结果集中的数据进行访问，其常用方法及使用说明见表 6-4。

表 6-4　ResultSet 接口的方法列表

方法名	功能说明
boolean absolute(int row)	将游标移动到第 row 条记录
boolean relative(int rows)	按相对行数（或正或负）移动游标
void beforeFirst()	将游标移动到结果集的开头（第一行之前）
boolean first()	将游标移动到结果集的第一行
boolean previous()	将游标移动到结果集的上一行
boolean next()	将游标从当前位置下移一行
boolean last()	将游标移动到结果集的最后一行
void afterLast()	将游标移动到结果集的末尾（最后一行之后）
boolean isAfterLast()	判断游标是否位于结果集的最后一行之后
boolean isBeforeFirst()	判断游标是否位于结果集的第一行之前
boolean isFirst()	判断游标是否位于结果集的第一行
boolean isLast()	判断游标是否位于结果集的最后一行
int getRow()	检索当前行编号
String getString(int x)	返回当前行第 x 列的值，类型为 String
int getInt(int x)	返回当前行第 x 列的值，类型为 int
Statement getStatement()	获取生成结果集的 Statement 对象
void close()	释放此 ResultSet 对象的数据库和 JDBC 资源
ResultSetMetaData getMetaData()	获取结果集的列的编号、类型和属性

ResultSet 维护指向当前数据行的游标。最初它位于第一行之前，每调用一次 next()方法，游标向下移动一行，从而按照从上至下的次序获取 ResultSet 数据行。同时提供一套 getXXX()方法对结果集中游标指向的行进行取值。在使用 getXXX()方法取值时，一定要注意数据库的字段数据类型和 Java 的数据类型之间的匹配。例如，对于数据库中的 int 或者 integer 的字段，对应的 Java 数据类型是 int，应该使用 ResultSet 类的 getInt()方法去读取。常用的 SQL 数据类型和 Java 数据类型之间的对应关系见表 6-5。

表 6-5　SQL 数据类型和 Java 数据类型之间的对应关系

SQL 数据类型	Java 数据类型	对应的方法
integer/int	int	getInt()
smallint	short	getShort()
float	double	getDouble()
double	double	getDouble()
real	float	getFloat()
varchar/char/varchar2	java.lang.String	getString()
boolean	boolean	getBoolean()
date	java.sql.Date	getDate()
time	java.sql.Time	getTime()
blob	java.sql.Blob	getBlob()
clob	java.sql.Clob	getClob()

在使用 getXXX()方法进行取值时，可通过列名或列号标识要获取数据的列。如：

```
String name = rs.getString("name");
String name = rs.getString(1);
```

ResultSet 中，列是从左至右编号的，并且从 1 开始。同时，用做 getXXX()方法的输入的列名不区分大小写。

用户不必关闭 ResultSet，当产生它的 Statement 关闭、重新执行或用于从多结果序列中获取下一个结果时，该 ResultSet 将被 Statement 自动关闭。

注意　在使用 getXXX()方法来获得数据库表中的对应字段的数据时，尽可能使用序列号参数，这样可以提高效率。

下述代码用于实现任务描述 6.D.2，使用 Statement 接口，实现对 Student 表的 CRUD 操作。

【描述 6.D.2】　StatementDemo.java

```
public class StatementDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String selectSql = "SELECT sno,name,age FROM student";
        String insertSql = "INSERT INTO student VALUES('95008','周瑜',19,1)";
        String updateSql = "UPDATE student SET age = age+1 WHERE sno=95008";
        String deleteSql = "DELETE FROM student WHERE sno=95008";
        try {
            // 通过工具类获取数据库连接
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            // 创建 Statment 对象
            Statement stmt = conn.createStatement();
            // 执行查询
            ResultSet rs = stmt.executeQuery(selectSql);
            System.out.println("查询成功！");
            while (rs.next()) {
                System.out.println("行 " + rs.getRow() + ":" + rs.getString(1)
                        + "," + rs.getString(2) + "," + rs.getString(3));
            }
            // 执行添加
            int count = stmt.executeUpdate(insertSql);
            System.out.println("添加" + count + "行！");
            // 执行修改
            count = stmt.executeUpdate(updateSql);
            System.out.println("修改" + count + "行！");
```

```
            // 执行删除
            count = stmt.executeUpdate(deleteSql);
            System.out.println("删除" + count + "行! ");
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
连接成功!
查询成功!
行 1:95001,张飞,23
行 2:95002,关羽,23
行 3:95003,貂蝉,23
行 4:95005,小乔,23
添加 1 行!
修改 1 行!
删除 1 行!
```

6.3.2 PreparedStatement 接口

PreparedStatement 接口是 Statement 接口的子接口，它继承了 Statement 的所有功能。PrepardStatement 接口有两大特点：

- 当使用 Statement 多次执行同一条 SQL 语句时，将会影响执行效率。此时可以使用 PreparedStatement 对象，如果数据库支持预编译，PreparedStatement 的对象中包含的 SQL 语句是预编译的，当需要多次执行同一条 SQL 语句时，可以直接执行预编译好的语句，其执行速度要快于 Statement 对象。
- PreparedStatement 的另一个特点可用于执行动态的 SQL 语句。所谓动态的 SQL 语句，就是可以在 SQL 语句中提供参数，这可大大提高程序的灵活性和执行效率。

可通过 Connection 的 prepareStatement()方法创建 PreparedStatement 对象。

创建用于 PreparedStatement 对象的动态 SQL 语句时，使用“？”作为动态参数的占位符。如：

```
String insertSql = "INSERT INTO student VALUES(?,?,?,?)";
PreparedStatement pstmt = conn.prepareStatement(insertSql);
```

在执行带参数的 SQL 语句前，必须对“？”进行赋值。PreparedStatement 接口中增添了

大量的 setXXX()方法，通过占位符的索引完成对输入参数的赋值，XXX 根据不同的数据类型选择，例如：

```
pstmt.setString(1, "95008");
```

下述代码用于实现任务描述 6.D.3，使用 PreparedStatement 接口实现对 Student 表的动态 CRUD 操作。

【描述 6.D.3】　PreparedStatementDemo.java

```
public class PreparedStatementDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String selectSql = "SELECT sno,name,age FROM student WHERE age > ?";
        String insertSql = "INSERT INTO student VALUES(?,?,?,?)";
        String updateSql = "UPDATE student SET age = age+1 WHERE sno=?";
        String deleteSql = "DELETE FROM student WHERE sno=?";
        try {
            // 通过工具类获取数据库连接
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            // 创建 Statment 对象
            PreparedStatement pstmt = conn.prepareStatement(selectSql);
            pstmt.setInt(1, 20);
            // 执行查询
            ResultSet rs = pstmt.executeQuery();
            System.out.println("查询成功！");
            while (rs.next()) {
                System.out.println("行 " + rs.getRow() + ":" + rs.getString(1)
                        + "," + rs.getString(2) + "," + rs.getString(3));
            }
            // 执行添加
            pstmt = conn.prepareStatement(insertSql);
            // 为占位符赋值
            pstmt.setString(1, "95008");
            pstmt.setString(2, "周瑜");
            pstmt.setInt(3, 19);
            pstmt.setString(4, "1");
            int count = pstmt.executeUpdate();
            System.out.println("添加" + count + "行！");
            // 执行修改
            pstmt = conn.prepareStatement(updateSql);
            pstmt.setString(1, "95008");
            count = pstmt.executeUpdate();
            System.out.println("修改" + count + "行！");
```

```
            // 执行删除
            pstmt = conn.prepareStatement(deleteSql);
            pstmt.setString(1, "95008");
            count = pstmt.executeUpdate();
            System.out.println("删除" + count + "行! ");
            // // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
连接成功!
查询成功!
行 1:95001,张飞,23
行 2:95002,关羽,23
行 3:95003,貂蝉,23
行 4:95005,小乔,23
添加 1 行!
修改 1 行!
删除 1 行!
```

6.3.3 CallableStatement 接口

存储过程是数据库中已经存在的 SQL 语句。在 JDBC 中执行数据库中的存储过程可以使用 CallableStatement 对象。

CallableStatement 接口继承了 Statement 接口和 PreparedStatement 接口，它具有两者的特点：可以处理一般的 SQL 语句，也可以处理输入参数，同时它还定义了 OUT（输出）参数及 INOUT（输入输出）参数的处理方法。

对象创建一个 CallableStatement 对象可以使用 Connection 类的 prepareCall()方法：

```
CallableStatement cstmt=conn.prepareCall("{call proc_name}");
```

其中，proc_name 是需要调用的存储过程名称。如果存储过程有参数，可使用占位符“?”代替，使用如下的方式调用：

```
CallableStatement cstmt=conn.prepareCall("{call proc_name(?,?)}");
```

其中，占位符“?”是 IN、OUT 还是 INOUT 取决于存储过程定义。CallableStatement 接口通过 setXXX()方法对 IN 参数进行赋值，对 OUT 参数，CallableStatement 提供方法进行类型注册和检索其值。

1. OUT 参数类型注册的方法

在执行一个存储过程之前，必须先对其中的 OUT 参数进行类型注册，注册方式如下：

```
registerOutParamenter(int index,int sqlType);
```

第一个参数为对应存储过程占位符，第二个参数为对应占位符的变量的类型，可以通过 java.sql.Types 的静态常量来指定，例如，Types.FLOAT、Types.INTEGER 等。如：

```
cstmt.registerOutParameter(2,java.sql.Types.VARCHAR);
```

当使用 getXXX()方法获取 OUT 参数的值时，XXX 对应的 Java 类型必须与所注册的 SQL 类型相符。

2. 查询结果的获取

由于 CallableStatement 允许执行带 OUT 参数的存储过程，所以它提供了完善的 getXXX()方法来获取 OUT 参数的值。除了 IN 参数与 OUT 参数外，还有一种 INOUT 参数。INOUT 参数具有其他两种参数的全部功能，可以用 setXXX()方法对参数值进行设置，再对这个参数进行类型注册，允许对此参数使用 getXXX()方法。执行完带此参数的 SQL 声明后，用 getXXX()方法可获取改变了的值。

如果存储过程有返回值，使用下面的方式调用：

```
CallableStatement cstmt=conn.prepareCall("{?=call proc_name(?,?)}");
```

注意　CallableStatement 一般用于执行存储过程，执行结果可能为多个 ResultSet、多次修改记录或两者都有。所以对 CallableStatement 一般调用方法 execute()执行 SQL 语句。

下述代码用于实现任务描述 6.D.4，使用 CallableStatement 实现各种情形存储过程的调用。

首先，基于 Student 表创建如下存储过程：

```
--添加学生信息
CREATE OR REPLACE PROCEDURE add_student(sno student.sno%TYPE,
name student.name%TYPE,age  student.age%TYPE,sex  student.sex%TYPE DEFAULT 1)
IS
BEGIN
INSERT INTO student
VALUES(sno,name,age,sex);
END;
--获取学生总数
CREATE OR REPLACE PROCEDURE get_count(
sum out number)
IS
BEGIN
SELECT count(*) INTO sum
```

```
FROM student ;
END;
--传递两个数，并分别通过参数 1 和参数 2 返回两个数的和与差
create or replace procedure add_sub(
num1 in out number,num2 in out number)
is
begin
num1 := num1+num2;
num2 := num1-num2-num2;
end;
```

【描述 6.D.4】 CallableStatementDemo.java

```
public class CallableStatementDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String callSql1 = "{call add_student(?,?,?,?)}";
        String callSql2 = "{call get_count(?)}";
        String callSql3 = "{call add_sub(?,?)}";
        try {
            // 通过工具类获取数据库连接
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            CallableStatement cstmt = null;
            // 调用带 IN 参数的存储过程
            cstmt=conn.prepareCall(callSql1);
            // 为 IN 参数赋值
            cstmt.setString(1, "95010");
            cstmt.setString(2, "赵云");
            cstmt.setInt(3, 22);
            cstmt.setString(4, "1");
            // 执行查询
            cstmt.execute();
            // 调用带 OUT 参数的存储过程
            cstmt=conn.prepareCall(callSql2);
            // 注册 OUT 参数的类型
            cstmt.registerOutParameter(1,java.sql.Types.INTEGER);
            cstmt.execute();
            int count = cstmt.getInt(1);
            System.out.println("总人数为:"+count);
            // 调用带 IN/OUT 参数的存储过程
            cstmt=conn.prepareCall(callSql3);
            // 为 IN 参数赋值
            cstmt.setInt(1, 10);
            cstmt.setInt(2, 5);
```

```
            // 注册 OUT 参数的类型
            cstmt.registerOutParameter(1,java.sql.Types.INTEGER);
            cstmt.registerOutParameter(2,java.sql.Types.INTEGER);
            cstmt.execute();
            // 通过参数索引获取返回值
            int sum = cstmt.getInt(1);
            int sub = cstmt.getInt(2);
            System.out.println("和:"+sum+",差:"+sub);
            // // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
总人数为:6
和:15,差:5
```

注意　在 JDBC3.0 及以上版本中，使用 CallableStatement 的 setXXX()/getXXX()方法来设置参数/取值的时候，既可以使用索引，也可以使用参数名称。JDBC2.0 及以下的版本，只能使用索引。

6.4　集元数据

所谓集元数据（Meta Data），就是有关数据库和表格结构的信息，如：数据库中的表、表的列、表的索引、数据类型、对 SQL 的支持程度等信息。

6.4.1　DatabaseMetaData 接口

DatabaseMetaData 接口主要是用来得到关于数据库的信息，如：数据库中所有表格的列表、系统函数、关键字、数据库产品名和数据库支持的 JDBC 驱动器名。DatabaseMetaData 对象是通过 Connection 接口的 getMetaData()方法创建的。

DatabaseMetaData 接口提供了大量获取信息的方法，这些方法可分为两大类：一类返回值为 boolean 型，多用以检查数据库或驱动器是否支持某项功能；另一类则用以获取数据库或驱动器本身的某些特征值。其常用方法及使用说明见表 6-6。

表 6-6　DatabaseMetaData 接口的常用方法列表

方法名	功能说明
boolean supportsOuterJolns()	检查数据库是否支持外部连接

（续表）

方法名	功能说明
boolean supportsStoredProcedures()	检查数据库是否支持存储过程
String getURL()	该方法的功能是返回用于连接数据库的 URL 地址
String getUserName()	该方法的功能是获取当前用户名
String getDatabaseProductName()	该方法的功能是获取使用的数据库产品名
String getDatabaseProductVersion()	该方法的功能是获取使用的数据库版本号
String getDriverName()	该方法的功能是获取用以连接的驱动器名称
String getProductVerslon()	该方法的功能是获取用以连接的驱动器版本号
ResultSet getTypeInfo()	该方法的功能是获取数据库中可能取得的所有数据类型的描述

下述代码用于实现任务描述 6.D.5，使用 DatabaseMetaData 获取当前数据库连接的相关信息。

【描述 6.D.5】 DatabaseMetaDataDemo.java

```
public class DatabaseMetaDataDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        try {
            // 通过工具类获取数据库连接
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            // 创建 DatabaseMetaData 对象
            DatabaseMetaData dmd = conn.getMetaData();
            System.out.println("数据库产品：" +dmd.getDatabaseProductName());
           System.out.println("数据库版本：" +dmd.getDatabaseProductVersion());
           System.out.println("驱动器：" +dmd.getDriverName());
           System.out.println("数据库 URL：" +dmd.getURL());
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
数据库产品：Oracle
数据库版本：Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options
驱动器：Oracle JDBC driver
数据库 URL：jdbc:oracle:thin:@localhost:1521:orc
```

6.4.2　ResultSetMetaData 接口

ResultSetMetaData 接口主要用来获取结果集的结构。例如，结果集的列的数量、列的名字等。可以通过 ResultSet 的 getMetaData()方法来获得对应的 ResultSetMetaData 对象。ResultSetMetaData 接口的常用方法及功能见表 6-7。

表 6-7　ResultSetMetaData 接口的方法列表

方法名	功能说明
int getColumnCount()	返回此 ResultSet 对象中的列数
String getColumnName(int column)	获取指定列的名称
int getColumnType(int column)	检索指定列的 SQL 类型
String getTableName(int column)	获取指定列的名称
int getColumnDisplaySize(int column)	指示指定列的最大标准宽度，以字符为单位
boolean isAutoIncrement(int column)	指示是否自动为指定列进行编号，这样这些列仍然是只读的
int isNullable(int column)	指示指定列中的值是否可以为 null
boolean isSearchable(int column)	指示是否可以在 where 子句中使用指定的列
boolean isReadOnly(int column)	指示指定的列是否明确不可写入

下述代码用于实现任务描述 6.D.6，使用 ResultSetMetaData 获取当前结果集的相关信息。

【描述 6.D.6】　ResultSetMetaDataDemo.java

```
public class ResultSetMetaDataDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String selectSql = "SELECT sno,name,age FROM student";
        try {
            // 通过工具类获取数据库连接
            Connection conn = DBUtil.getConnection();
            System.out.println("连接成功！");
            // 创建 Statment 对象
            Statement stmt = conn.createStatement();
            // 执行查询
            ResultSet rs = stmt.executeQuery(selectSql);
            ResultSetMetaData rsmd = rs.getMetaData();
            System.out.println("总共有：" + rsmd.getColumnCount()+"列");
            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                System.out.println("列" + i + ":" + rsmd.getColumnName(i) + ","
                        + rsmd.getColumnTypeName(i) + "("
                        + rsmd.getColumnDisplaySize(i) + ")");
            }
            // 关闭连接
            DBUtil.closeConnection(conn);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
总共有：3 列
列 1:SNO,VARCHAR2(5)
列 2:NAME,VARCHAR2(10)
列 3:AGE,NUMBER(22)
```

6.5 事务操作

数据库中一些操作的集合通常是一个独立单元，而事务就是构成单一逻辑工作单位的操作集合。已提交事务是指成功执行完毕的事务，未能成功完成的事务称为中止事务，对中止事务造成的变更需要进行撤销处理，称为事务回滚。

事务具有 ACID 四个特性：

- **原子性（A）**：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行。
- **一致性（C）**：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。
- **隔离性（I）**：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。
- **持久性（D）**：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障。

6.5.1 事务

在 JDBC 中，事务操作默认是自动提交的。也就是说，一条对数据库的更新操作成功后，系统将自动调用 commit()方法提交，否则将调用 rollback()方法来回滚。

为了能将多个 JDBC 语句组合成一个操作单元，以保证数据的一致性，可以禁止自动提交，之后把多个数据库操作的表达式作为一个事务，在操作完成后调用 commit()来进行整体提交，倘若其中一个表达式操作失败，可以在异常捕获时调用 rollback()进行回滚。

利用 Connection 对象的 setAutoCommit()方法，可以开启或者关闭自动提交方式，它接受一个 boolean 类型的参数，如果为 false，表示关闭自动提交，如果为 true，则表示打开自动提交。可以使用 Connection 对象的 getAutoCommit()方法来检查自动提交方式是否打开。如

果将自动提交功能关闭，就可以调用 Connection 的 commit()方法来提交所有更新或者用 rollback()方法来取消更新。

下述代码用于实现任务描述 6.D.7，通过更新操作演示在 JDBC 中的事务控制。

【描述 6.D.7】　TransactionDemo java

```
public class TransactionDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String insertSql1 = "INSERT INTO student VALUES('95011','张三',19,1)";
        String insertSql2 = "INSERT INTO student VALUES('95012','李四',20,0)";
        String insertSql3 = "INSERT INTO student VALUES('95011','王五',18,1)";
        Connection conn = null;
        try {
            conn = DBUtil.getConnection();
            Statement stmt = conn.createStatement();
            // 获取并记录事务提交状态
            boolean autoCommit = conn.getAutoCommit();
            conn.setAutoCommit(false);
            // 添加批处理语句
            stmt.executeUpdate(insertSql1);
            stmt.executeUpdate(insertSql2);
            // 由于 Student 表的主键约束，下述语句将抛出异常
            stmt.executeUpdate(insertSql3);
            // 如果顺利执行则在此提交
            conn.commit();
            // 恢复原有事务提交状态
            conn.setAutoCommit(autoCommit);
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            // 如果出现异常，回滚
            if(conn!=null){
                try{
                    conn.rollback();
                }
                catch(SQLException se){
                    se.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }
}
```

上述代码执行期间，由于 Student 表的主键限制，将会在执行：

```
stmt.executeUpdate(insertSql3);
```

语句时，抛出主键约束异常，从而使程序入口转到 catch 语句中，通过调用：

```
conn.rollback();
```

撤销所有操作。对于运行结果，可通过查询 Student 表的数据进行验证。

6.5.2 保存点

JDBC 同时还支持保存点操作，通过保存点，可以更好地控制事务回滚。

下述代码用于实现任务描述 6.D.8，在 JDBC 中通过设置保存点，控制事务的部分回滚。

【描述 6.D.8】 SavepointDemo.java

```
public class SavepointDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String insertSql1 = "INSERT INTO student VALUES('95011','张三',19,1)";
        String insertSql2 = "INSERT INTO student VALUES('95012','李四',20,0)";
        Connection conn = null;
        Savepoint s1 = null;
        try {
            conn = DBUtil.getConnection();
            Statement stmt = conn.createStatement();
            // 获取并记录事务提交状态
            boolean autoCommit = conn.getAutoCommit();
            conn.setAutoCommit(false);
            // 添加批处理语句
            stmt.executeUpdate(insertSql1);
            // 设置保存点
            s1 = conn.setSavepoint();
            stmt.executeUpdate(insertSql2);
            // 回滚保存点
            if(true){
                conn.rollback(s1);
            }
            // 如果顺利执行则在此提交
            conn.commit();
            // 恢复原有事务提交状态
            conn.setAutoCommit(autoCommit);
            // 关闭连接
            DBUtil.closeConnection(conn);
```

```
        } catch (Exception e) {
            // 如果出现异常，回滚
            if (conn != null) {
                try {
                    conn.rollback();
                } catch (SQLException se) {
                    se.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }
}
```

上述代码在执行第二条数据插入操作前通过如下语句设置了一个名为 s1 的保存点：

```
s1 = conn.setSavepoint();
```

并在程序执行期间，通过逻辑判断，使用语句：

```
conn.rollback(s1);
```

对 s1 保存点到当前语句之间的操作进行了部分回滚。通过查询 Student 表中的数据可以验证，第一条数据插入操作顺利完成，第二条操作并未奏效。

注意　如果使用 Oracle10g 提供的 class12.jar 中的驱动时，运行期间会抛出类似“java.lang.AbstractMethodError: oracle.jdbc.driver.T4CConnection”的异常，此时可以将驱动程序包更换为 Oracle 10g 自带的 ojdbc14.jar 包。



小结

通过本章的学习，学生应该能够学会：

- JDBC 是 Java 应用与数据库通信的基础。
- JDBC 包含一组类与接口，用于连接到任何数据库。
- JDBC 访问数据库的一般步骤如下。
 - **01** 加载 JDBC 驱动程序。
 - **02** 建立数据库连接。
 - **03** 创建 Statement 对象。
 - **04** 执行 SQL 语句。
 - **05** 处理返回结果。
 - **06** 关闭创建的对象。

- JDBC 通过 ResultSet 维持查询结果集，并提供游标进行数据操作。
- 通过 Statement 实现静态 SQL 查询。
- 使用 PreparedStatement 实现动态 SQL 查询。
- 使用 CallableSatement 实现存储过程的调用。
- DatabaseMetaData 接口用于得到关于数据库的信息。
- ResultSetMetaData 接口主要用来获取结果集的结构。
- JDBC 默认的事务提交模式是自动提交。
- 通过 setAutoCommit()方法控制自动提交模式，使用 rollback()方法实现事务回滚。



1. 以下代码行的功能是_____。

```
Class.forName("Sun.jdbc.odbc.JdbcOdbcDriver");
```

A. 为服务器数据库加载驱动程序　　B. 建立与指定数据库的连接
C. 创建 ResultSet 对象　　D. 访问表中数据

2. 为维护不同数据库所创建的驱动器的列表，使用以下_____选项。
A. JDBC 驱动器管理程序　　B. JDBC-ODBC 桥接
C. JDBCODBC.dll　　D. 库例程
3. 以下代码片段来自 Java 源文件：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connectioncon=DriverManager.getConnection("jdbc:odbc:MyDataSource","user1","""
);
Statement stat=con.createStatement();
result=stat.executeQuery("Select * from Publishers");
```

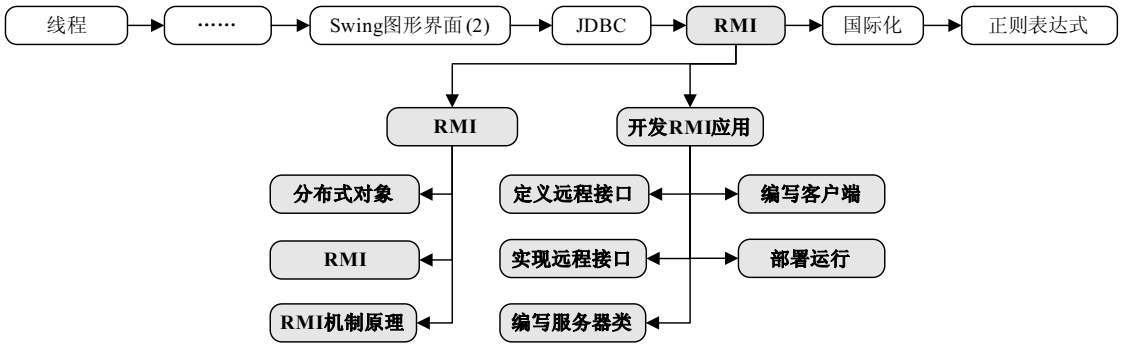
为得到 result 中的列数，以下代码正确的是_____。
A. ResultMetaData rsmd=DatabaseMetaData.getMetaData();
int columns=rsmd.getColumnCount();
B. ResultSetMetaData rsmd=new ResultSetMetaData(result);
Int columns=rsmd.getColumnCount();
C. ResultSetMetaData rsmd=result.getMetaData();
Int columns=rsmd.getColumnCount();
D. DatabaseMetaData md=result.getMetaData();
Int columns=md.getColumnCount();
4. JDBC 连接 Oracle 数据库的驱动类是_____。
5. JDBC 连接 Oracle 数据库的连接字符串是_____。
6. 简述 JDBC 访问数据库的步骤。
7. 使用 JDBC 查询 Scott 用户的 emp 表中的所有信息。

第 7 章　RMI

本章目标

- 了解 RMI 概念及架构
- 了解 RMI 的机制原理
- 掌握 RMI 开发流程
- 掌握开发、部署 RMI 的流程

学习导航



任务描述

【描述 7.D.1】

定义远程接口，提供加、减、乘、除的服务功能。

【描述 7.D.2】

定义远程接口实现类，提供加、减、乘、除的具体服务。

【描述 7.D.3】

针对 7.D.1 和 7.D.2 的远程接口和远程接口实现类，编写服务器程序。

【描述 7.D.4】

编写客户端，实现 RMI 远程服务的调用。

【描述 7.D.5】

部署并运行 RMI 应用。

7.1　RMI 概述

随着软件技术和网络技术的发展，网络应用无论从硬件设备还是软件应用上都表现为多元化，终端之间的通信也不仅局限于同种设备或同种语言。

7.1.1　分布式对象

在传统的 C/S 模型中，客户端与服务器进行通信需要遵循一定的规范，一般是：客户端将请求信息按照规范翻译成某种中间传输格式，然后将请求发送给服务器；服务器接收到数据后，将信息进行“翻译”，通过运算得到响应信息，再将响应信息格式化并传递给客户；然后客户端解析响应信息并将它呈现给用户。

对于此种数据传输，如果手动实现此过程，首先要解决的就是：需设计一种专用的传输格式，同时为需要传输的数据与该传输格式之间的转换而编码。其处理过程如图 7-1 所示。

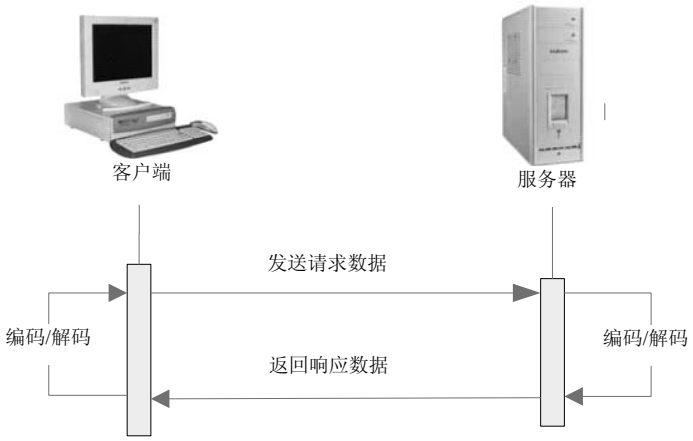


图 7–1　C/S 数据传输模型

产生上述问题的关键在于，客户端和服务器端可能不在同一个 JVM（Java 虚拟机）上，而且客户端和服务器端的实现语言可能不同。对于此种“异域”问题，可以通过代理（proxy）技术解决。

对于“异域”问题，使用代理技术解决的一般步骤是：在客户端安装一个服务器代理，客户端通过此服务器代理进行常规的方法调用，而此服务器代理负责与服务器进行通信；同样，在服务器端安装一个客户端代理，该客户端代理负责与客户端通信，服务器通过客户端代理与客户进行通信。通过两种代理的安装，客户端与服务器的通信转换为两个代理的通信。对于开发人员而言，可以通过最简单的方法调用实现 C/S 通信功能而不必关心代理是如何通信的。基于代理的数据传输模型如图 7-2 所示。

常用的基于代理的通信技术有如下几种：

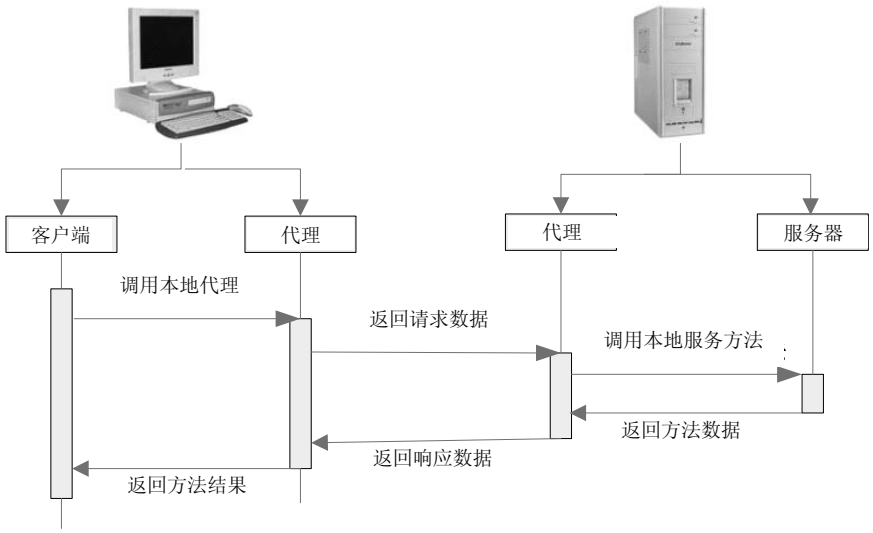


图 7-2　基于代理的数据传输模型

- RMI：Java 远程方法调用，是基于 Java 的分布式对象之间的调用方法。
- CORBA：通用对象请求代理架构，基于 IIOP（Internet Inter-ORB）协议之间的对象通信，支持各种语言编写的对象之间的调用。
- SOAP：简单对象访问协议，是基于 XML 传输格式的协议，更适合于 Web 架构的系统。

其中，CORBA 和 SOAP 是完全独立于语言的，客户端和服务器程序可以由 Java、C、C++或其他语言编写。

7.1.2　RMI

Java RMI（Remote Method Invocation，远程方法调用）是用 Java 在 JDK1.1 中实现的，是 Enterprise JavaBeans 的支柱，是建立分布式 Java 应用程序的方便途径。

它大大增强了 Java 开发分布式应用的能力。Java 作为一种风靡一时的网络开发语言，其巨大的威力就体现在它强大的开发分布式网络应用的能力上，而 RMI 就是开发百分之百纯 Java 的网络分布式应用系统的核心解决方案之一。Java RMI 则支持存储于不同地址空间的程序级对象之间彼此进行通信，实现远程对象之间的无缝远程调用。RMI 目前使用 Java 远程消息交换协议 JRMP（Java Remote Messaging Protocol）进行通信。JRMP 是专为 Java 的远程对象制定的协议。因此，Java RMI 具有 Java 的“Write Once，Run Anywhere”的优点，是分布式应用系统的百分之百纯 Java 解决方案。用 Java RMI 开发的应用系统可以部署在任何支持 JRE（Java Run Environment，Java 运行环境）的平台上。但由于 JRMP 是专为 Java 对象制定的，因此，RMI 对于用非 Java 语言开发的应用系统的支持不足，不能与用非 Java 语言书写的对象进行通信。

7.1.3　RMI 机制原理

RMI 应用程序通常包括两个独立的程序：服务器程序和客户机程序。典型的服务器应用程序将创建多个远程对象，并调用注册服务程序以使名字与远程对象相关联，使这些远程对象能够被引用，然后等待客户机调用这些远程对象的方法。而客户机在服务器上的注册服务程序中用远程对象的名字查找一个或多个远程对象的引用，然后调用远程对象的方法。

RMI 为服务器和客户机进行通信和信息传递提供了一种机制。在与远程对象的通信过程中，RMI 使用标准机制：Stub 和 Skeleton。远程对象的 Stub 担当远程对象的客户本地代表或代理人角色。调用程序将调用本地 Stub 的方法，而本地 Stub 将负责执行对远程对象的方法调用。在 RMI 中，远程对象的 Stub 与该远程对象所实现的远程接口集相同。调用 Stub 的方法时将执行下列操作：

01 初始化与包含远程对象的远程虚拟机的连接。
02 对远程虚拟机的参数进行编组（写入并传输）。
03 等待方法调用结果。
04 解编（读取）返回值或返回的异常。
05 将值返回给调用程序。

为了向调用程序展示比较简单的调用机制，Stub 将参数的序列化和网络级通信等细节隐藏了起来。

在远程虚拟机中，每个远程对象都可以有相应的 Skeleton（在 JDK1.5 环境中不必使用 Skeleton）。Skeleton 负责将调用分配给实际的远程对象实现。它在接收方法调用时执行下列操作：

01 解编（读取）远程方法的参数。
02 调用实际远程对象实现上的方法。
03 将结果（返回值或异常）编组（写入并传输）给调用程序。

Stub 和 Skeleton 由 rmic 编译器生成。

利用 RMI 编写分布式对象应用程序需要完成以下工作：

- 定位远程对象。应用程序可使用两种机制中的一种得到对远程对象的引用。它既可用 RMI 的简单命名工具 rmiregistry 来注册它的远程对象，也可以将远程对象引用作为常规操作的一部分来进行传递和返回。
- 与远程对象通信。远程对象间通信的细节由 RMI 处理，对于程序员来说，远程通信看起来就像标准的 Java 方法调用。
- 动态类加载。当一个 Java 对象作为远程方法的参数或返回值，传递给另一个程序时，该程序必须拥有此远程对象的类文件。而且，RMI 将提供必要的机制，既可以加载对象的代码又可以传输对象的数据。

在 RMI 分布式应用系统中，服务器与客户机之间传递的 Java 对象必须是可序列化的对象。不可序列化的对象不能在对象流中进行传递。对象序列化扩展了核心 Java 输入/输出类，同时也支持对象。对象序列化支持把对象编码及将通过它们可访问到的对象编码变成字节流；同时，它也支持流中对象图形的互补重构造。序列化用于轻型持久性和借助于套接字或远程方法调用（RMI）进行的通信。

序列化 Java 对象的目的是：提供一种简单但可扩充的机制，以序列化方式维护 Java 对象的类型及安全属性；具有支持编组和解编的扩展能力以满足远程对象的需要；具有可扩展性以支持 Java 对象的简单持久性；只有在自定义时，才需对每个类提供序列化自实现；允许对象定义其外部格式。

7.2 开发 RMI

编写 Java RMI 分布式应用程序的步骤主要包括以下几步：

01 定义远程接口。远程接口中定义了需要远程调用的方法。

02 实现远程接口。远程接口实现类必须实现远程接口中定义的远程方法。

03 编写和实现服务器类。注册远程对象服务。

04 编写使用远程服务的客户端程序。在该类中使用 java.rmi.Naming 中的 lookup()方法获得对远程对象的引用，依据需要调用该引用的远程方法，其调用方式和对本地对象方法的调用相同。

7.2.1 定义远程接口

在 RMI 中的远程接口中声明需要远程调用的方法，远程对象是实现远程接口的类的实例。在需要创建一个远程对象的时候，通过传递一个接口来隐藏基层的实施细节，客户通过接口句柄发送消息即可。

RMI 的远程接口具有如下特点：

- 远程接口必须声明为 public。
- 远程对象扩展 java.rmi.Remote 接口。
- 每个方法还必须抛出 java.rmi.RemoteException 异常。
- 任何作为参数或返回值传送的远程对象的数据类型必须声明为远程接口类型，而不是实现类。

下述代码用于实现任务描述 7.D.1，定义远程接口，提供加、减、乘、除的服务功能。

【描述 7.D.1】 Calculator.java

```
public interface Calculator extends Remote {
```

```
    // 加法操作
    public long add(long a, long b) throws java.rmi.RemoteException;
    // 减法操作
    public long sub(long a, long b) throws java.rmi.RemoteException;
    // 乘法操作
    public long mul(long a, long b) throws java.rmi.RemoteException;
    // 除法操作
    public long div(long a, long b) throws java.rmi.RemoteException;
}
```

注意　这个接口继承自 Remote，每一个定义的方法都必须抛出一个 RemoteException 异常对象。

7.2.2　实现远程接口

远程接口实现类必须扩展远程对象 java.rmi.UnicastRemoteObject 类，并实现所定义的远程接口。远程接口实现类必须实现远程接口中定义的远程方法。这个类也可以含有附加的方法，但客户只能使用远程接口中的方法。

必须为远程接口实现类定义构造函数，这个构造函数通过 super()调用基础类的构造函数，并激活 UnicastRemoteObject 中的代码完成 RMI 的连接和远程对象的初始化。基础类构造函数可能会抛出 java.rmi.RemoteException，所以此构造函数必须抛出 java.rmi.RemoteException 异常。

下述代码用于实现任务描述 7.D.2，定义远程接口实现类，提供加、减、乘、除的具体服务。

【描述 7.D.2】 CalculatorImpl.java

```
public class CalculatorImpl extends UnicastRemoteObject implements Calculator {
    // 需提供一个显式的构造函数，并且要抛出一个 RemoteException 异常
    public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }
    // 实现加法操作
    public long add(long a, long b) throws java.rmi.RemoteException {
        return a + b;
    }
    // 实现减法操作
    public long sub(long a, long b) throws java.rmi.RemoteException {
        return a - b;
    }
    // 实现乘法操作
    public long mul(long a, long b) throws java.rmi.RemoteException {
```

```
        return a * b;
    }
    // 实现除法操作
    public long div(long a, long b) throws java.rmi.RemoteException {
        return a / b;
    }
}
```

CalculatorImpl 类通过继承 UnicastRemoteObject 去连接 RMI 系统。事实上并不一定要这样做，如果一个类不是从 UnicastRmeoteObject 上继承的，可以通过调用它的 exportObject() 方法去连接到 RMI。

7.2.3 编写服务器类

服务器类一般完成两个功能：

- 提供 main 方法（也可以是另一个类）。
- 提供远程对象注册服务。

RMI 类库提供了自举注册服务（bootstrap registry service）来定位服务器对象。服务器程序使用自举注册服务来注册对象，然后客户端就可以获取通过这些对象的 Stub。通过给自举注册服务提供一个对象的引用和一个名字，就可以注册一个服务器对象。

下述代码用于实现任务描述 7.D.3，针对 7.D.1 和 7.D.2 的远程接口和远程接口实现类，编写服务器程序。

【描述 7.D.3】 CalculatorServer.java

```
public class CalculatorServer {
    public CalculatorServer() {
        try {
            // 声明远程对象
            Calculator cal = new CalculatorImpl();
            // 声明全局注册表
            Context context = new InitialContext();
            // 注册远程对象
            context.bind("rmi:CalculatorService", cal);
            System.out.println("Server started...");
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }

    public static void main(String args[]) {
        new CalculatorServer();
```

```
    }
}
```

上述服务器类中，使用语句：

```
Context context = new InitialContext();
```

获取全局注册表，并通过调用 Context 类的 bind()方法实现服务对象注册功能。此时 RMI 命名服务已经被整合到 JNDI（Java 命名与目录信息）服务中了。在 JDK1.3 或更低的版本中，使用的是单独的 RMI 命名服务，上述注册代码可调整为：

```
// 声明远程对象
Calculator cal = new CalculatorImpl();
// 注册远程对象
Naming.bind("rmi:CalculatorService", cal);
```

注意　在注册远程对象时，提供的服务别名在同一个注册表中不能重复。

7.2.4　编写客户端

客户端程序请求 RMI 注册服务对象，并通过返回的服务器对象调用其功能方法。RMI 中通过调用 Context 类的 lookup()方法实现服务查找功能，lookup()方法需要接受一个 RMI 的 URL 用于指定查找的路径。此 URL 的格式如下：

```
rmi://rmi_host:port/service_name
```

其中：

rmi_host 指定 RMI 服务器的地址，默认为 localhost。

port 指定 RMI 服务的端口，默认为 1099。

service_name 指定需要查找的服务的名字。

下述代码用于实现任务描述 7.D.4，编写客户端，实现 RMI 远程服务的调用。

【描述 7.D.4】 CalculatorClient.java

```
public class CalculatorClient {
    public static void main(String[] args) throws NamingException {
        try {
            // 声明全局注册表
            Context context = new InitialContext();
            // 查找 RMI 服务中的服务对象
            Calculator cal = (Calculator) context
                    .lookup("rmi://localhost/CalculatorService");
            // Calculator c = (Calculator)
            // Naming.lookup("rmi://localhost/CalculatorService");
```

```
            System.out.println(cal.add(10, 5));
            System.out.println(cal.sub(10, 3));
            System.out.println(cal.mul(8, 8));
            System.out.println(cal.div(15, 3));
        } catch (RemoteException re) {
            re.printStackTrace();
        } catch (ArithmeticException ae) {
            ae.printStackTrace();
        }
    }
}
```

当调用 Context 类的 lookup()方法时，如果根据指定的 RMI 地址搜索到远程对象，将返回远程对象的引用，此时需强制进行类型转换；如果搜索不到，将抛出 RemoteException 异常。

在 JDK1.3 或更低的版本中，使用的是单独的 RMI 命名服务，上述查找代码可调整为：

```
Calculator c = (Calculator)Naming.lookup("rmi://localhost/CalculatorService");
```

有一点需要特殊指明：对于 RMI 应用，通常情况下，服务器程序在不断地变化，新的类及接口不断地被添加到服务器，此时就需要客户端动态加载类。在 Java 中，只要需要从另一个地方加载代码，就需要一个安全管理器。

在编写 RMI 客户端程序时，可以通过 RMISecurityManager 类用以控制动态加载存根的行为，可以通过以下语句安装：

```
System.setSecurityManager(new RMISecurityManager);
```

如果所有的类和存根在本地都是可用的，安全管理器就可有可无了。

7.2.5 部署运行

实现了服务器和客户机的程序后，就是编译和运行该 RMI 系统了。其步骤有：

01 使用 javac 编译远程接口类，远程接口实现类和客户机程序。
02 使用 rmic 编译器生成实现类的 stub 和 skeleton（JDK1.5 以前的版本需要此步骤）。
03 启动 RMI 注册服务程序 rmiregistry。
04 启动服务器端程序。
05 启动客户机程序。

下述操作实现任务描述 7.D.5，部署并运行 RMI 应用。

1. 编译程序

在 Windows 下，通过命令行模式进入 RMI 源文件所在文件夹，使用 javac 命令编译远程

接口、实现类、服务器程序、客户端程序等源文件。

```
javac *.java
```

针对上述应用，编译后会生成 4 个 class 文件。

2. 启动 RMI 注册表

RMI 注册表程序用于接收 RMI 服务注册，并为客户端程序提供服务定位，必须在服务器端程序之前启动。在命令行模式下，通过如下语句进行启动：

```
start rmiregistry
```

使用 start 命令将弹出一个新的命令行窗口，如果启动完毕会弹出一个命令行窗口，RMI 注册表程序将一直运行，直到关闭此命令行窗口。

3. 启动服务器程序

对于服务器程序的运行，其必要的类文件为远程接口类、远程接口实现类、服务器程序及相关辅助类等。

在命令行窗口中，键入如下语句启动服务器程序：

```
start java CalculatorServer
```

如果服务器启动成功，将会在新的命令行窗口中输出下述语句：

```
Server started...
```

此时服务器已随时等待客户端调用了。

4. 启动客户端程序

对于客户端程序的运行，其必要的类文件为远程接口类、客户端程序及相关辅助类等。

在命令行窗口中，键入如下语句启动客户端程序。

```
java CalculatorClient
```

此时客户端会得到如下运行结果。

```
15
7
64
5
```

程序执行通过。

为更好地理解 RMI 的运行，读者可将服务器程序和客户端程序及其所依赖的类文件分别复制到不同的文件夹中，按照运行次序再次启动、测试。

7.2.6 注意事项

1. 在 JDK1.4 之前的版本上运行 RMI

如果使用的是JDK1.4或更低的版本，需要使用rmic编译器生成实现类的stub和skeleton。对于上述应用执行如下语句：

```
rmic CalculatorImpl
```

将会生成名为 CalculatorImpl_stub 的类文件，此文件即 Stub 文件，需要随客户端程序和服务器程序一块部署。

2. 安全策略

RMI 的安全机制将在服务端发生作用。要允许客户连接 RMI 注册表及获取服务对象，需为 RMI 提供一个策略文件，如：

```
grant{
    permission java.security.AllPermission;
}
```

它定义了最简单的安全策略，允许任何人做任何事。

```
grant{
    permission java.net.SocketPermission "*:2000-3000", "connect";
}
```

上述策略规定了允许应用程序建立任何网络连接，只要端口号在2000~3000范围内即可。

在客户端程序中，通过将 java.security.policy 属性设为策略文件的名字，就可以指示安全管理器去读取该策略文件，如：

```
System.setProperty("java.security.policy","my.policy");
```

上述代码假设策略文件的文件名为“my.policy”。

小结

通过本章的学习，学生应该能够学会：

- RMI 是便于编程员调用跨网络环境的分布构件。
- RMI 是百分之百纯 Java 的网络分布式应用系统的解决方案。
- RMI 是一种规格说明，能使一台 Java 虚拟机（JVM）调用位于另一台 Java 虚拟机上的对象。
- RMI 的机制是客户端调用本地代理，本地代理与远程代理联系，远程代理调用服务

器方法并返回结果。

- 远程对象是实现远程接口的类的实例，它扩展了 java.rmi.Remote 接口。
- 远程接口实现类必须扩展远程对象 java.rmi.UnicastRemoteObject 类，并且为远程接口实现类定义构造方法。
- 创建 RMI 应用的步骤为：

01 定义远程接口——远程接口类。
02 定义和实现服务器中的远程方法——服务器类。
03 定义与实现客户——客户类。
04 编译源文件。
05 创建安全策略。
06 启动 RMI 远程注册表。
07 启动服务器。
08 启动客户端。

练习

1. 以下命令的结果将是_____。

```
rmic -d c:\MyRMI AccountServerImpl
```

A. 在驱动器 C 的 MyRMI 目录中将产生 stub 和 skeleton
B. 该类路径将被 MyRMI 覆盖，并在当前目录中将产生 stub 和 skeleton
C. 在当前目录中将产生 stub 和 skeleton，其 bootstrap 路径将为 c:\MyRMI
D. 在当前目录中将产生 stub 和 skeleton，所产生的.java 文件保留在 c:\MyRMI 中

2. 现有一个使用 RMI 注册表的默认端口的 RMI 应用，如果给出以下命令来启动 RMI 注册表，代码中将需作什么变动？_____。

```
start rmiregistry 1099
```

A. 改变 rebind()方法
B. 改变 lookup()方法
C. 改变 rebind() 和 lookup()
D. 不改变任何代码

3. 如何用 RMI 传递远程对象_____。

A. 复制传递
B. 引用传递
C. 如果它是静态的，复制传递

D. 如果它是短暂的，复制传递

4. UnicastRemoteObject 的构造符的作用是_____？

A. 导出远程对象

B. 创建远程对象

C. 执行远程对象

D. 实例化远程对象

5. RMI 架构由_____、_____和_____三层构成。

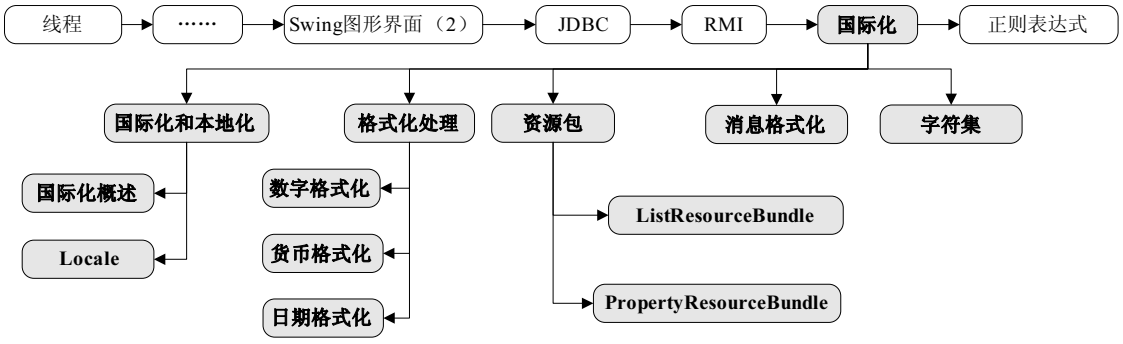
编写远程 RMI 应用程序：服务器提供两个数的乘积功能，编写服务器类和客户类的代码。

第8章 国 际 化

本章目标

- 理解国际化和本地化的概念
- 掌握 Locale 对象的使用
- 学会使用 NumberFormat 进行数字、货币格式化
- 学会使用 DateFormat 格式化日期和时间
- 学会使用 MessageFormat 格式化消息字符串
- 掌握编辑、定位资源包的方法
- 掌握如何访问属性文件
- 理解常用的字符集之间的区别

学习导航



任务描述

【描述 8.D.1】

使用 Locale 获取本地信息，并演示 Locale 类的使用。

【描述 8.D.2】

获取当前系统能够支持的本地化信息。

【描述 8.D.3】

使用 NumberFormat 实现数字格式化处理。

【描述 8.D.4】

使用 NumberFormat 实现货币格式化处理。

【描述 8.D.5】

使用 DateFormat 实现日期时间格式化处理。

【描述 8.D.6】

使用 SimpleDateFormat 实现日期时间格式化处理。

【描述 8.D.7】

使用 ListResourceBundle 实现资源格式化处理。

【描述 8.D.8】

使用 PropertyResourceBundle 实现资源格式化处理。

【描述 8.D.9】

使用 MessageFormat 实现消息格式化处理。

8.1　国际化和本地化

随着 Internet 时代和全球化时代的到来，万维网（World Wide Web）的迅猛发展推动了国际业务的发展，软件全球化也成为其中的一条支流。国际化（多国语言）是应用服务得以推广的基础。

8.1.1　国际化概述

软件全球化，首先就要使程序能支持多国语言。如果应用是面向多种语言的，编程时就不得不设法解决国际化问题，包括操作界面的风格问题、提示和帮助语言的版本问题、界面定制个性化问题等。

国际化是设计和编写应用程序以便可以在全球或多国环境中使用的过程。国际化程序能够支持不同的语言及不同格式的日期、时间、货币和其他值，而不必修改软件。这通常涉及“软编码”或将文本组件同程序代码分离并且可能涉及可插入代码模块。

国际化（简写为 I18N，表示 internationalization 一词开始的 I 和最后的 N 之间有 18 个字母）在软件设计阶段就应该使其具备支持多种语言的功能。这样，当需要在应用中添加对一种新语言的支持时，就不需要对已有的软件重构。

本地化则是设计和编写能够处理特定区域、国家或地区、语言、文化、企业或政治环境的应用程序的过程。从某种意义上说，为特定地区编写的所有应用程序都本地化了，这些应用程序大多数只支持一种语言环境。本地化（localization）通常缩写为 L10N。

可以这样理解国际化，看下述代码：

```
public static void main(String[] args) {
    System.out.println("Hello.");
    System.out.println("How are you.");
    System.out.println("Goodbye.");
}
```

上述代码会向懂英语的人输出英文问候消息，而面对一个不懂英语的日本人进行同样的问候时，则不得不重新调整代码中的问候语句。

可以这样理解，国际化意味着一个软件可同时支持多种语言，而本地化需要为不同用户提供不同版本的软件，其区别可参考图 8-1。

从图 8-1 可以看出，实现国际化的思路就是抽取具备“语言”特性的描述到资源文件中，需要时再根据实际情况关联即可。

由于 Java 语言具有平台无关、可移植性好等特点，并且提供了强大的类库，且 Java 语言本身采用 Unicode 编码，所以使用 Java 语言可以方便地实现国际化。从设计角度来说，只要把程序中与语言和文化有关的部分分离出来，加上特殊处理，就可以部分解决国际化问题。在界面风格的定制方面，把可以参数化的元素，如字体、颜色等抽取到资源文件或数据库中，

以便为用户提供友好的界面；如果某些部分包含无法参数化的元素，则不得不分别设计，通过“硬编码”来解决具体问题。

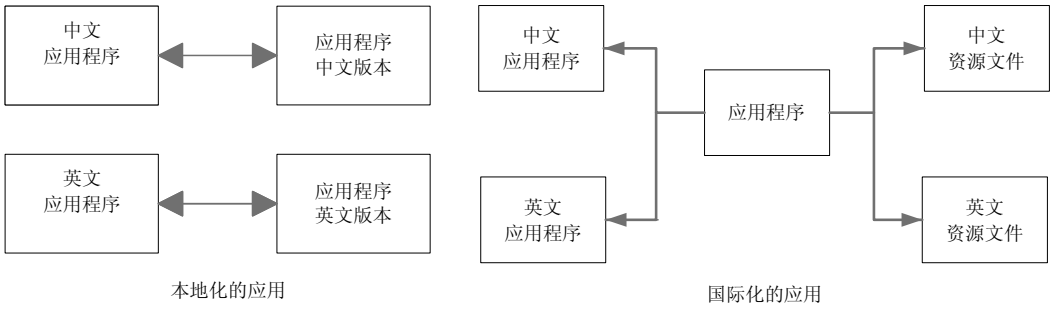


图 8-1　本地化与国际化的区别

在 Java 中，为解决国际化问题，可能利用到的主要的类都是由 java.util 包提供的。该类包中相关的类有 Locale、ResourceBundle、ListResourceBundle、PropertyResourceBundle 等。

8.1.2　Locale 类

在 Java 中，语言环境（locale）仅仅是一个标识符，Locale 类是用来标识本地化消息的重要工具类。该类包含对主要地理区域的地域化特征的封装。其特定对象表示某一特定的地理、政治或文化区域。通过设定 Locale 为特定的国家或地区，提供符合当地文化习惯的字体、符号、图标和表达格式。一个 Locale 实例代表一种特定的语言和地区，可以通过 Locale 对象中的信息来输出其对应语言和地区的时间、日期、数字等格式。其常用方法及使用说明见表 8-1。

表 8-1　Locale 类的方法列表

方法名	功能说明
Local(String language)	构造 language 指定的语言的 Locale 对象
Local(String language,String country)	构造 language 指定的语言和 country 指定的国家（地区）的 Locale 对象
String getCountry()	返回国家（地区）代码
String getDisplayCountry()	返回国家（地区）名称
String getLanguage()	返回语言代码
String getDisplayLanguage()	返回语言名称
Static Locale getDefault()	获取当前系统信息的对应的 Locale 对象
Static void setDefault(Locale new)	重新设置默认的 Locale 对象

根据构造方法，要声明一个完整的 Locale 对象，至少要提供 language 和 country 参数，而这两个参数的指定是有限制的，例如，要定义一个中国大陆的 Locale 对象，则 language 应取“zh”，country 应取“CN”，需执行：

```
Locale locale = new Locale("zh","CN");
```

在这里 language 指定的参数参照 ISO—639 标准，它以两个小写字母来表示某种语言。完整的语言编码列表可以参考：http://www.unicode.org/unicode/onlinedat/languages.html。表 8-2 列举了本章使用的 ISO—639 语言列表。

表 8–2　ISO—639 语言列表

语言	英文名称	编码
汉语	Chinese	zh
英语	English	en
日语	Japanese	ja
德语	German	de

country 指定的参数参照 ISO—3166 标准，它以两个大写字母来表示国家（地区）。完整的国家（地区）编码列表可以参考：http://www.unicode.org/unicode/onlinedat/countries.html。表 8-3 列举了本章使用的 ISO—3166 国家（地区）列表

表 8–3　ISO—3166 国家（地区）列表

国家（地区）	英文名称	编码
中国	China	CN
英国	Great Britain	GB
日本	Japan	JP
美国	United States	US
德国	Germany	DE

为方便记忆和使用，在 Locale 类中还通过静态常量定义了一些常用的 Locale 对象，如：Locale.CHINA、Locale.JAPAN 等，而 Locale. SIMPLIFIED_CHINESE 则只设置了语言并没有指定区域信息。可以通过执行如下语句：

```
System.out.println(Locale.CHINA);
System.out.println(Locale.US);
```

获取静态常量中保存的语言信息和国家（地区）信息。

下述代码用于实现任务描述 8.D.1，使用 Locale 获取本地信息，并演示 Locale 类的使用。

【描述 8.D.1】 LocaleDemo.java

```
public class LocaleDemo {
    public static void main(String[] args) {
        // 获取默认的 Locale 信息
        Locale locale = Locale.getDefault();
        // 语言代码
        System.out.println("Language       : " + locale.getLanguage());
        // 国家代码
```

```
        System.out.println("Country        : " + locale.getCountry());
        // 语言显示名称
        System.out.println("DisplayLanguage : " + locale.getDisplayLanguage());
        // 国家显示名称
        System.out.println("DisplayCountry  : " + locale.getDisplayCountry());
        System.out.println("locale : " + locale);
        Locale newLocale = new Locale("en", "US");
        // 重设默认 Locale 信息
        Locale.setDefault(newLocale);
        locale = Locale.getDefault();
        System.out.println("Language       : " + locale.getLanguage());
        System.out.println("Country        : " + locale.getCountry());
        System.out.println("DisplayLanguage : " + locale.getDisplayLanguage());
        System.out.println("DisplayCountry  : " + locale.getDisplayCountry());
        System.out.println("locale : " + locale);
    }
}
```

执行结果如下：

```
Language       : zh
Country        : CN
DisplayLanguage : 中文
DisplayCountry  : 中国
locale : zh_CN
Language       : en
Country        : US
DisplayLanguage : English
DisplayCountry  : United States
locale : en_US
```

在使用 Locale 时，指定的语言及国家（地区）信息需要在本地系统的支持下才能通过，可以调用 Locale.getAvailableLocales()方法来取得当前系统支持的本地化信息。

下述代码用于实现任务描述 8.D.2，获取当前系统能够支持的本地化信息。

【描述 8.D.2】 LocaleDemo2.java

```
public class LocaleDemo2 {
    public static void main(String[] args) {
        // 获取系统支持的语言和国家（地区）列表
        Locale locales[]=Locale.getAvailableLocales();
        for (Locale locale : locales ){
                System.out.println(locale);
        }
    }
}
```

8.2　格式化处理

全世界使用的日期有各种各样的顺序和符号，数字和货币也是如此。在编程语言中，如果靠自己手工来处理这些情形，几乎不可能。借助于 Java 提供的 Locale 类，却能够处理每个受支持语言环境的所有的不同格式。依赖 Locale 类，Java 中提供了一系列的格式器（Formatter）来完成数字、货币和日期的格式化。

8.2.1　数字格式化

在不同的国家（地区），数字表示是不一样的，如在中国的“8,888.8”，在德国却表示为“8.888,8”，因此对数字的表示要根据不同的 Locale 来格式化。

在 java.text 包中提供了一个 NumberFormat 类，用于完成对数字、百分比进行格式化和字符串对象解析。NumberFormat 提供了大量的静态方法用于获取使用指定 Locale 对象封装的 NumberFormat 实例。其常用的方法及使用说明见表 8-4。

表 8–4　NumberFormat 的方法列表

方法名	功能说明
static NumberFormat getNumberInstance()	返回与当前系统信息相关的默认的数字格式器对象
static NumberFormat getNumberInstance(Locale l)	返回指定 Locale 为 l 的数字格式器对象
static NumberFormat getPercentInstance()	返回与当前系统信息相关的默认的百分比格式器对象
static NumberFormat getPercentInstance(Locale l)	返回指定 Locale 为 l 的百分比格式器对象
static NumberFormat getCurrencyInstance()	返回与当前系统信息相关的默认的货币格式器对象
static NumberFormat getCurrencyInstance (Locale l)	返回指定 Locale 为 l 的货币格式器对象
String format(double number)	将数字 number 格式化为字符串返回
Number parse(String source)	将指定的字符串解析为 Number 对象

下述代码用于实现任务描述 8.D.3，使用 NumberFormat 实现数字格式化处理。

【描述 8.D.3】 NumberFormatDemo.java

```
public class NumberFormatDemo {
    public static void main(String args[]) {
        // 需要格式化的数据
        double value = 123456.789;
        // 设定三个 Locale
        Locale cnLocale = new Locale("zh", "CN");
        Locale usLocale = new Locale("en", "US");
        // 德国
        Locale deLocal3 = new Locale("de", "DE");
        NumberFormat dNf = NumberFormat.getNumberInstance();
```

```
        NumberFormat pNf = NumberFormat.getPercentInstance();
        // 得到三个 local 对应的 NumberFormat 对象
        NumberFormat cnNf = NumberFormat.getNumberInstance(cnLocale);
        NumberFormat usNf = NumberFormat.getNumberInstance(usLocale);
        NumberFormat deNf = NumberFormat.getNumberInstance(deLocal3);
        // 将上边的 double 数据格式化输出
        System.out.println("Default Percent Format:" + pNf.format(value));
        System.out.println("Default Number Format:" + dNf.format(value));
        System.out.println("China Number Format:" + cnNf.format(value));
        System.out.println("United Number Format:" + usNf.format(value));
        System.out.println("German Number Format:" + deNf.format(value));
        String str1 = "123456.789";
        String str2 = "123456.789a";
        String str3 = "a123456.789a";
        try {
            System.out.println(dNf.parse(str1).doubleValue());
            System.out.println(dNf.parse(str2).doubleValue());
            // 下述语句抛出异常
            System.out.println(dNf.parse(str3).doubleValue());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

执行结果如下：

```
Default Percent Format:12,345,679%
Default Number Format:123,456.789
China Number Format:123,456.789
United Number Format:123,456.789
German Number Format:123.456,789
123456.789
123456.789
java.text.ParseException: Unparseable number: "a123456.789a"
```

上述代码中声明了中文、英文和德文的三个 Locale 对象，并使用相应的 NumberFormat 对指定的数据格式化输出。

parse()方法的返回类型是 Number，如果给定的数字文本格式不正确，该方法会抛出 ParseException 异常，例如，要转换的字符串为“a123456.789a”则会抛出异常。但是，任何跟在数字之后的字符将被忽略，如“123456.789a”则会顺利转换。

8.2.2　货币格式化

NumberFormat 除了能对数字、百分比格式化外，还可以对货币数据格式化。货币格式化通常是在钱数前面加上类似于“￥”、“$”的货币符号，来区分货币类型。

可以使用 NumberFormat 的静态方法 getCurrencyInstance()方法来获取格式器。

下述代码用于实现任务描述 8.D.4，使用 NumberFormat 实现货币格式化处理。

【描述 8.D.4】 CurrencyFormatDemo.java

```
public class CurrencyFormatDemo {
    public static void main(String args[]) {
        // 需要格式化的数据
        double value = 123456.784;
        // 设定 Locale
        Locale cnLocale = new Locale("zh", "CN");
        Locale usLocale = new Locale("en", "US");
        // 得到 local 对应的 NumberFormat 对象
        NumberFormat cnNf = NumberFormat.getCurrencyInstance(cnLocale);
        NumberFormat usNf = NumberFormat.getCurrencyInstance(usLocale);
        // 将上边的 double 数据格式化输出
        System.out.println("China Currency Format:" + cnNf.format(value));
        System.out.println("United Currency Format:" + usNf.format(value));
    }
}
```

执行结果如下：

```
China Currency Format:￥123,456.78
United Currency Format:$123,456.78
```

通过执行结果可以看出，在以货币格式输出时，在数据前面添加了相应的货币符号，并且在人民币和美元的表示中，都精确到了“分”，确保数据有意义。

8.2.3　日期格式化

就像数字和货币，不同的国家（地区）也有不同的日期和时间格式。如，中文环境下的日期格式为“xxxx 年 xx 月 xx 日”，而英文环境下可能就是“yyyy-mm-dd”。因此对日期和时间也需要根据不同的 Locale 来格式化。

在 Java 中，日期和时间的格式化是通过 DateFormat 来完成的，其使用方式与 NumberFormat 类似，DateFormat 常用方法及使用说明见表 8-5。

表 8-5　DateFormat 的方法列表

方法名	功能说明
static DateFormat getDateInstance()	返回默认样式的日期格式器
static DateFormat getDateInstance(int style)	返回默认指定样式的日期格式器
static DateFormat getDateInstance(int style, Locale aLocale)	返回默认指定样式和 Locale 信息的日期格式器
static DateFormat getTimeInstance()	返回默认样式的时间格式器
static DateFormat getTimeInstance(int style)	返回默认指定样式的时间格式器
static DateFormat getTimeInstance(int style, Locale aLocale)	返回默认指定样式和 Locale 信息的时间格式器
static DateFormat getDateTimeInstance()	返回默认样式的日期时间格式器
static DateFormat getDateTimeInstance(int dateStyle, int timeStyle)	返回默认指定样式的日期时间格式器
static DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale aLocale)	返回默认指定样式和 Locale 信息的日期时间格式器

表 8-5 提及的日期（时间）样式（dateStyle，timeStyle），是 DateFormat 中定义好的静态常量，用于控制输出日期、时间的显示形式，常用的样式控制有：

- DateFormat.FULL：在 zh_CN 的 Locale 下，此格式的日期格式取值类似于“2010 年 3 月 22 日 星期一”，时间格式取值类似于“下午 01 时 26 分 14 秒 CST”。
- DateFormat.LONG：在 zh_CN 的 Locale 下，此格式的日期格式取值类似于“2010 年 3 月 22 日”，时间格式取值类似于“下午 01 时 26 分 14 秒”。
- DateFormat.DEFAULT：在 zh_CN 的 Locale 下，此格式的日期格式取值类似于“2010-3-22”，时间格式取值类似于“13:26:14”。
- DateFormat.SHORT：在 zh_CN 的 Locale 下，此格式的日期格式取值类似于“10-3-22”，时间格式取值类似于“下午 1:26”。

下述代码用于实现任务描述 8.D.5，使用 DateFormat 实现日期和时间的格式化处理。

【描述 8.D.5】 DateFormatDemo.java

```
public class DateFormatDemo {
    public static void main(String[] args) {
        Date now = new Date();
        Locale cnLocale = new Locale("zh", "CN");
        Locale usLocale = new Locale("en", "US");
        System.out.println("中文格式：");
        print(now, cnLocale);
        System.out.println("英文格式：");
        print(now, usLocale);
    }
    public static void print(Date date, Locale locale) {
```

```
        // 得到对应 Locale 对象的日期格式化对象
        DateFormat df1 = DateFormat.getDateTimeInstance(DateFormat.FULL,
                DateFormat.FULL, locale);
        DateFormat df2 = DateFormat.getDateTimeInstance(DateFormat.LONG,
                DateFormat.LONG, locale);
        DateFormat df3 = DateFormat.getDateTimeInstance(DateFormat.DEFAULT,
                DateFormat.DEFAULT, locale);
        DateFormat df4 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
                DateFormat.SHORT, locale);
        // 格式化日期输出
        System.out.println(df1.format(date));
        System.out.println(df2.format(date));
        System.out.println(df3.format(date));
        System.out.println(df4.format(date));
    }
}
```

执行结果如下：

```
中文格式:
2010 年 3 月 22 日 星期一 下午 01 时 35 分 33 秒 CST
2010 年 3 月 22 日 下午 01 时 35 分 33 秒
2010-3-22 13:35:33
10-3-22 下午 1:35
英文格式:
Monday, March 22, 2010 1:35:33 PM CST
March 22, 2010 1:35:33 PM CST
Mar 22, 2010 1:35:33 PM
3/22/10 1:35 PM
```

在 Java 中还提供了更加简便的日期格式器，即 DateFormat 的子类 SimpleDateFormat，通过使用 SimpleDateFormat，可以更加灵活地格式化日期（时间）。

SimpleDateFormat 的使用很简单，通过预定义好模式字符构造特定的模式串，然后根据模式串来创建 SimpleDateFormat 格式器对象，从而通过此格式器完成指定日期时间的格式化。例如：“D”表示一年中的第几天，“d”表示一月中的第几天，“E”代表星期中的第几天等，其他可以使用的模式字符可参看 Java 提供的 API 帮助文档，表 8-6 列举了本章使用的日期模式字符。

表 8-6　部分日期模式字符

模式字符	功能说明
D	一年中的第几天
d	一月中的第几天
E	星期中的第几天

（续表）

模式字符	功能说明
y	年
H	小时（0~23）
h	小时（0~11），使用 AM/PM 区分上下午
M	月份
m	分钟
S	毫秒
s	秒

通过模式字符可以构建控制日期、时间格式的模式串，在 zh_CN 的 Locale 下自定义模式串及其对应的日期、时间格式，示例如表 8-7 所示。

表 8–7　日期模式串示例

格式串	输出实例
yyyy.MM.dd G 'at' HH:mm:ss	2010.03.22 公元 at 13:57:47
h:mm a	1:58 下午
yyyy 年 MM 月 dd 日 HH 时 mm 分 ss 秒	2010 年 03 月 22 日 13 时 50 分 02 秒
EEE, d MMM yyyy HH:mm:ss	星期一, 22 三月 2010 13:58:52
yyyy-MM-dd HH:mm:ss	2010-03-22 13:50:02

注意　如果需要在模式串中使用的字符（字符串）不被 SimpleDateFormat 解释，可以在模式串中将其用单引号括起来。

下述代码用于实现任务描述 8.D.6，使用 SimpleDateFormat 实现日期时间的格式化处理。

【描述 8.D.6】 SimpleDateFormatDemo.java

```
public class SimpleDateFormatDemo {
    public static void main(String[] args) {
        Date now = new Date();
        SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println(sdf1.format(now));
        SimpleDateFormat sdf2 =
        new SimpleDateFormat("yyyy年MM月dd日 HH时mm分ss秒");
        System.out.println(sdf2.format(now));
        SimpleDateFormat sdf3 =
        new SimpleDateFormat("现在是 yyyy年 MM 月 dd 日，是今年的第 D 天");
        System.out.println(sdf3.format(now));
    }
}
```

执行结果如下：

```
2010-03-22 14:05:29
2010 年 03 月 22 日 14 时 05 分 29 秒
现在是 2010 年 03 月 22 日，是今年的第 81 天
```

注意　SimpleDateFormat 一般不用于国际化处理，其提出是为了以特定模式输出日期（时间），更方便本地化使用。

8.3　资源包

使用 Locale 对象可以方便地实现日期、时间、数字等的格式化，但是在实际开发中，一个应用可能会有大量的消息字符串、按钮标签和菜单等内容也需要针对不同的 Locale 翻译成不同的语言。为了能方便地实现根据不同的语言选配不同的字符串，可以把这些消息抽取出来，定义为外部资源，这样就可以将程序和资源轻松分离。这些外部资源一般都采用“键-值”对形式提供，此处的“键”作为提取所需数据的标识，同一项目的键要保持一致，而“值”则是针对不同 Locale 对应的语言翻译，每个 Locale 都对应自己的资源包。

当定义一个国际化的应用时，会有很多资源包。每个包都是与本地 Locale 相关的属性文件或相关类的集合。如想新增一种 Locale 支持，只需要增加对应的资源包即可。在 Java 内部资源包有统一的命名规则：

- 包名_语言_国家（地区），定义和国家（地区）相关的资源。
- 包名_语言，定义与语言相关的资源。

上述命名规则中，包名可以任意取，只要符合 Java 的命名规范即可。如定义中文对应的资源包可以为“resource_zh”，针对我国大陆地区则可以用“resource_zh_CN”，对于我国台湾地区可以取名为“resource_zh_TW”，而对于同一个应用的英文资源命名应为“resource_en”。

定义好资源包后，系统会根据 Locale 对象中的国家（地区）和语言的信息来查找对应的资源包，其查找次序为：

- 包名_当前 Locale 语言_当前 Locale 国家（地区）。
- 包名_当前 Locale 语言。
- 包名。

例如，当前 Locale 对象为（“zh”,” CN”），则系统会查找“包名_zh_CN”资源文件，如果没有则查找“包名_zh”资源文件，如果还不成功，就会加载“包名”对应的资源，如果再次失败，加载资源失败，应用将抛出 MissingResourceException 异常。

8.3.1 ListResourceBundle

java.util 包中提供了 ResourceBundle 抽象类来封装特定的资源信息，可以使用其提供的 getString 或 getObject 方法方便地获取指定的资源。其常用方法及使用说明见表 8-8。

表 8-8 ResourceBundle 的方法列表

方法名	功能说明
static ResourceBundle getBundle(String baseName)	根据指定的 baseName 路径，返回默认 Locale 的 ResourceBundle 对象
static ResourceBundle getBundle(String baseName, Locale locale)	根据指定的 baseName 路径和 locale 信息，返回 ResourceBundle 对象
String getString(String key)	根据指定的键 key 返回其对应的字符串值
Object getObject(String key)	根据指定的键 key 返回其对应的对象值

为使用 ResourceBundle，必须实现 getKeys()和 handleGetObject()方法，工作较烦锁。在此基础上，Java 提供了 ResourceBundle 的子类 ListResourceBundle，只需提供 getContents()方法即可，此方法用于返回当前资源类所封装的资源信息，为二维数组形式。

下述代码用于实现任务描述 8.D.7，使用 ListResourceBundle 实现资源的格式化处理。

【描述 8.D.7】 MyResource_en_US.java

```
/*
 * 默认资源类，英文
 */
public class MyResource_en_US extends ListResourceBundle {
    private final Object my_data[][] = {
            { "text", "This is english character" }, { "hello", "Hello!" },
            { "bye", "Bye!" } };
    // 需重写此方法,返回数据集
    protected Object[][] getContents() {
        return my_data;
    }
}
```

【描述 8.D.7】 MyResource_zh_CN.java

```
/*
 * 中文资源类
 */
public class MyResource_zh_CN extends ListResourceBundle {
    private final Object my_data[][] = { { "text", "这是中文文本" },
            { "hello", "你好!" }, { "bye", "再见!" } };
    // 需重写此方法,返回数据集
    public Object[][] getContents() {
```

```
        return my_data;
    }
}
```

上述代码定义了英文和中文的资源类，其都继承自 ListResourceBundle 类，并对getContents()方法进行了重写，而且资源类的命名需要遵循“包名_语言_国家”的原则。下面是测试代码。

【描述 8.D.7】 ListResourceBundleDemo.java

```
public class ListResourceBundleDemo {
    public static void main(String[] args) {
        Locale defaultLocale = Locale.getDefault();
        print("com.haiersoft.ch08.MyResource", defaultLocale);
        Locale enLocale = new Locale("en", "US");
        print("com.haiersoft.ch08.MyResource", enLocale);
        Locale cnLocale = new Locale("zh", "CN");
        print("com.haiersoft.ch08.MyResource", cnLocale);
    }
    // 根据传递的资源文件名和 Locale 对象信息打印相应的数据信息
    public static void print(String resource, Locale locale) {
        System.out.println("---------");
        System.out.println("Locale : " + locale);
        ResourceBundle bundle = ResourceBundle.getBundle(resource, locale);
        System.out.print("text : " + bundle.getObject("text"));
        System.out.print(",hello: " + bundle.getObject("hello"));
        System.out.println(",bye  : " + bundle.getObject("bye"));
    }
}
```

执行结果如下：

```
Locale : zh_CN
text : 这是中文文本,hello: 你好!,bye  : 再见!
---------
Locale : en_US
text : This is english character,hello: Hello!,bye  : Bye!
---------
Locale : zh_CN
text : 这是中文文本,hello: 你好!,bye  : 再见!
```

上述代码分别使用默认的、英文和中文本地化消息取得对应的消息资源。有一点值得注意，在提供资源类时，需要指定类路径的全称，如 com.haiersoft.ch08.MyResource。

8.3.2 PropertyResourceBundle

ListResourceBundle 不但可以封装字符串信息，还可以封装其他对象信息，只需要在getContents()方法中提供对象的二维数组即可。如果资源文件中只包含字符串信息，则可以使用 PropertyResourceBundle。

PropertyResourceBundle 也是 ResourceBundle 的子类，其只能处理字符串格式的资源。使用 PropertyResourceBundle 的好处是，所有的资源信息都以“键/值”对形式，放在属性文件中，免去了编译的过程，方便调整和修改。PropertyResourceBundle 使用的属性文件扩展名必须为“.properties”，其文件命名也需遵循资源包命名原则。

使用 PropertyResourceBundle 还有一点要注意。Java 使用 Unicode 编码，而属性文件是文本文件，文本文件一般以系统的本地编码格式编码（中文系统下为 GB2312），这样就可能产生非 Unicode 编码，致使在应用中取出的资源为乱码。为防止出现这个问题，对于此类的属性资源文件应进行转码操作，将所有的编码转换成对应的 Unicode 编码。可以运行 Java 自带的命令 native2ascii 进行转码，命令格式为：

```
native2ascii   encoding gb2312 源文件.properties 目标文件.properties
```

其中源文件为本地编码的文本文件，目标文件为转换好的编码文件。

下述代码用于实现任务描述 8.D.8，使用 PropertyResourceBundle 实现资源格式化处理。

首先定义中文资源的属性文件。

【描述 8.D.8】 MyResource_zh_temp.properties

```
text=这是中文文本
hello=你好!
bye=再见!
```

通过命令行格式进入中文资源文件所在文件夹，运行 native2ascii 命令，将中文资源内容转换为 Unicode 编码格式：

```
native2ascii -encoding gb2312
MyResource_zh_temp.properties MyResource_zh_CN.properties
```

这里目标文件的命名必须遵循“包名_语言_国家（地区）”原则。编码后的属性文件如下：

【描述 8.D.8】 MyResource_zh_CN.properties

```
text=\u8fd9\u662f\u4e2d\u6587\u6587\u672c
hello=\u4f60\u597d!
bye=\u518d\u89c1!
```

英文属性文件本身为 ASCII 编码，无需编码处理，其内容如下：

【描述 8.D.8】 MyResource_en_US.properties

```
text=This is english character
hello=Hello!
bye=Bye!
```

下述代码是测试代码：

【描述 8.D.8】 PropertyResourceBundleDemo.java

```
public class PropertyResourceBundleDemo {
    public static void main(String[] args) {
        Locale defaultLocale = Locale.getDefault();
        print("properties.MyResource", defaultLocale);
        Locale enLocale = new Locale("en", "US");
        print("properties.MyResource", enLocale);
        Locale cnLocale = new Locale("zh", "CN");
        print("properties.MyResource", cnLocale);
    }
    // 根据传递的资源文件名和 Locale 对象信息打印相应的数据信息
    public static void print(String resource, Locale locale) {
        System.out.println("---------");
        System.out.println("Locale : " + locale);
        ResourceBundle bundle = ResourceBundle.getBundle(resource, locale);
        System.out.print("text : " + bundle.getObject("text"));
        System.out.print(",hello: " + bundle.getObject("hello"));
        System.out.println(",bye  : " + bundle.getObject("bye"));
    }
}
```

执行结果如下：

```
Locale : zh_CN
text : 这是中文文本,hello: 你好!,bye  : 再见!
---------
Locale : en_US
text : This is english character,hello: Hello!,bye  : Bye!
---------
Locale : zh_CN
text : 这是中文文本,hello: 你好!,bye  : 再见!
```

注意　在使用资源文件时，只需指明资源文件所在的类路径全称即可，资源文件必须放置到某个包路径下，如本例的资源文件放在 properties 包下。

8.4　消息格式化

对于固定消息（常量文本）的应用，可以使用 ResourceBundle 处理。但要处理动态消息（变量文本）时，例如：要在用户每次登录系统的时候向其提示当前系统日期和时间，此条

信息的日期和时间部分是不固定的，而且还要根据用户的本地化消息输出不同的格式。

java.text 包中提供了 MessageFormat 类，可以实现消息的动态处理。此类接收一个带占位符的字符串，然后根据实际情况使用实际的值来替换字符串中的占位符。占位符的格式如下：

```
{n[,formatType][,formatStyle]}
```

其中，n 代表占位符的索引，取值是 0~9，formatType 表示格式类型，可以标识数字、日期、时间，formatStyle 表示格式的样式。常用占位符类型和样式见表 8-9。

表 8–9　占位符类型和样式的方法列表

类型	样式	功能说明
number	integer	整数类型
	currency	货币类型
	percent	百分比类型
	#.##	小数类型
date	full	完整格式日期
	long	长格式日期
	medium	中等格式日期
	short	短格式日期
time	full	完整格式时间
	long	长格式时间
	medium	中等格式时间
	short	短格式时间

不同于其他 Format 类，MessageFormat 对象是使用构造方法创建的，使用 MessageFormat 的一般步骤是：

01 构造字符串，其中动态变化的部分使用占位符代替，最多可以设置 10 个占位符，每个占位符可以多次重复出现。

02 使用上一步构造的字符串作为 MessageFormat 构造方法的参数，构造 MessageFormat 对象。

03 调用 MessageFormat 对象的 setLocale()方法设置它的 Locale 属性。

04 创建一个对象数组，并按照占位符的索引填充此数组。

05 将对象数组传递给 MessageFormat 的 format()方法。

下述代码用于实现任务描述 8.D.9，使用 MessageFormat 实现消息的格式化处理。

【描述 8.D.9】 MessageFormat1.java

```
public class MessageFormat1 {
    public static void main(String[] args) {
```

```
        // 使用占位符构造模式串
        String pattern = "{0},你好! 现在是 {1}";
        // 构造 MessageFormat 对象
        MessageFormat formatter = new MessageFormat(pattern);
        Locale locale = Locale.getDefault();
        formatter.setLocale(locale);
        // formatter.applyPattern(pattern);
        Date now = new Date();
        // 构造模式串所需的对象数组
        Object[] msgParams = { "张三", now };
        // 根据传递的参数，对应替换模式串中的占位符
        System.out.println(formatter.format(msgParams));
    }
}
```

执行结果如下：

```
同学,你好! 现在是  10-3-22 下午 6:42
```

向 format()方法传递的值必须放到 object[]数组中，MessageFormat 对象根据数组的下标值来对应占位符的数字进行替换，如本例中，占位符{0}被“同学”替换，{1}被“now”对象替换。实际应用中，还可以定义占位符的样式，即占位符后面可以跟一个类型和一种风格。

下述代码用于实现任务描述 8.D.10，为占位符设置样式，使用 MessageFormat 实现消息的格式化处理。

【描述 8.D.10】 MessageFormat2.java

```
public class MessageFormat2 {
    public static void main(String[] args) {
        // 对同一个占位符进行不同的格式化
        String pattern = "{0} ,你好!今天是{1,date,long},现在是{1,time,hh:mm:ss}";
        MessageFormat formater = new MessageFormat(pattern);
        Locale locale = Locale.getDefault();
        formater.setLocale(locale);
        Date now = new Date();
        Object[] msgs = { "同学", now };
        System.out.println(formater.format(msgs));
    }
}
```

执行结果如下：

```
同学 ,你好!今天是 2010 年 3 月 22 日,现在是 06:49:13
```

上述代码中，两次用到占位符{1}，因为程序中需要的数据来自同一个 Date 对象，但指明了不同的类型和风格，这样在程序执行时还是用“now”对象来替换所有占位符{1}，而不管模式串中有多少个相同的占位符。另外，在第三个占位符中使用了时间格式化串“hh:mm:ss”，更多格式信息可参考前面介绍的 SimpleDateFormat。

结合前面介绍的资源文件，可以将消息文本写进资源文件中，从而更方便、更灵活地实现国际化应用程序。

注意 在使用资源文件时，只需指明资源文件所在的类路径全称即可，资源文件必须放置到某个包路径下，如本例的资源文件放在 properties 包下。

8.5 字符集

1. ASCII

ASCII 编码是美国标准信息交换码，这种编码方式针对的是英文字符。ASCII 编码使用一个字节对字符进行编码，而且字节的最高位都为 0，因此 ASCII 编码的字符集大小是 128 个。由于英文字母仅有 26 个，再加上其他一些常用符号，总大小也不会超过 128 个，例如，字符“a”被编码为 0x61，字符“b”被编码为 0x62，等等。

2. ISO 8859—1

ISO 8859—1 是西欧语系国家通用的字符集编码，ISO 8859—1 使用一个字节对字符进行编码，编码值范围是 0x00~0xFF。其中，0x00~0x1F 用做控制字，0x20~0x7F 表示字母、数字和符号这些图形字符，0xA0~0xFF 作为附加部分使用。由于 ASCII 编码只使用了一个字节中的低 7 位，编码范围仅为 0~127，虽然可以容纳英文字符和其他的一些符号，但是却不能包含除英文以外的其他西欧语言的字母，因此 ASCII 编码在西欧国家并不通用。针对这个问题，ISO 在 ASCII 编码的基础上进行了扩充，制定了 ISO 8859—1 编码，ISO8859—1 编码使用了一个字节的全部 8 位，编码范围是 0~255，能包含西欧语系的所有字母和符号。

3. GB2312、GBK 和 BIG 5

GB2312 码是中华人民共和国国家汉字信息交换使用码，全称《信息交换使用汉字编码字符集——基本集》，由国家标准总局发布。GB2312 收录了简化汉字、符号、字母、日文假名等共计 7445 个字符，其中汉字占 6763 个。GB2312 的编码范围是 0x2121~0x777E，与 ASCII 有重叠，通常方法是将 GB 码的两个字节的最高位置 1 区别。

GBK 是 GB2312—80 的扩展，包含了 20902 个汉字，编码范围是 0x8140~0xFEFE。GB18030—2000（GBK2K）在 GBK 的基础上增加了藏、蒙等少数民族的字符，GBK2K 从根本上解决了字位不够、字形不足的问题。现在的 PC 平台必须支持 GB18030，对嵌入式产品暂不作要求，所以手机、MP3 一般只支持 GB2312。

BIG 5 码被称为大五码，是中国港台地区使用的字符编码方式，共计收入 13868 个字。

从 ASCII、GB2312、GBK 到 GB18030，这些编码方法是向下兼容的，即同一个字符在

这些方案中总是有相同的编码，后面的标准支持更多的字符。在这些编码中，英文和中文可以统一地处理。

4. Unicode

ISO（国际标准化组织）将全世界所有的符号进行统一编码，称为 Unicode 编码。Unicode 编码的字符占用两个字节（也有 4 字节的）的大小，对于 ASCII 码表示的字符，Unicode 只是简单地在 ASCII 码原来的一个字节码值上增加一个所有位全为 0 的字节。Unicode 使用两个字节编码，因此能表示的字符集最大为 65536，另外 Unicode 中还保留了两千多个数值未用于字符编码。由于 Unicode 编码的空间有限，只能包含各个地区常用的字符而非所有字符，因此，在相当长的一段时间里，本地化字符编码和 Unicode 编码将共存。

定长编码便于计算机处理（注意 GB2312/GBK 不是定长编码），而 Unicode 又可以用来表示所有字符，所以在很多软件内部是使用 Unicode 编码来处理的，如 Java。

5. UTF–8 和 UTF–16

UTF-8 和 UTF-16 编码仍然属于 Unicode 编码，它们是在 Unicode 编码基础上进行了转换或扩展。例如在 Windows XP 和 2000 操作系统中，Unicode 编码指的就是 UTF-16 编码。

UTF-8 编码是将 Unicode 编码中不同范围的字符采用不同的字节进行编码，对于 ASCII 编码的字符仍使用一个字节进行编码，UTF-8 编码完全兼容 ASCII 编码。与 Unicode 相比较，UTF-8 编码使得英文文档的占用空间减小了一半，因此 UTF-8 颇受英语系国家的青睐。除此之外，UTF-8 编码中不会出现值为 0x0000 的数据，这样避免了和某些程序语言产生冲突，而 UTF-8 编码的补充位使得数据能够被方便地检测出在传输过程中是否发生了错误。

UTF-16 编码在 Unicode 的基础上进行了一些细节上的扩充，增加了对 Unicode 编码没有包括的字符的表示方式。Unicode 是 UTF-16 的子集，UTF-16 对 Unicode 的扩充并没有影响 Unicode 编码中的原有字符。Unicode 编码将 0xD800~0xDFFF 区间的数值保留，被称为代理区间，区间共包含 2048 个数值，其中 0xD800~0xD6FF 是高半代理区，0xDC00~0xDFFF 是低半代理区。UTF-16 编码就是在 Unicode 编码基础上利用代理区扩充字符编码的机制。UTF-16 编码从两个区域分别取一个编码，组成一个 4 字节的代理对来表示一个编码字符，就能够在 Unicode 基础上扩充 1024×1024 个字符。UTF-16 足够用来编码全球的所有字符。微软从 Windows 2000 开始支持 UTF-16 编码。

6. Little–Endian 和 Big–Endian

在不同体系的计算机系统中，不同的 CPU 处理多字节字符在内存中存储的顺序是不同的。如果数据存储是低字节在前，高字节在后，这种存储方式被称为 Little-Endian（小尾）；如果数据存储是高字节在前，低字节在后，这种存储方式被称为 Big-Endian（大尾）。

小结

通过本章的学习，学生应该能够学会：

- Java 是一个全面支持国际化的语言，使用 Unicode 处理所有字符串。
- Java 通过类 Locale 设定语言及国家（地区）。
- NumberFormat 用于进行数字、货币格式化。
- DateFormat、SimpleDateFormat 用于格式化日期和时间。
- MessageFormat 用于格式化消息字符串。
- 在国际化应用程序中可以事先将信息资源包装在资源包中，程序根据 Locale 定位资源包内容，从而实现资源和程序的分离。
- ListResourceBundle 用于实现对象类型的资源格式化处理。
- PropertyResourceBundle 用于处理资源文件。
- GB2312 码是中华人民共和国国家汉字信息交换使用码。
- GBK 码是 GB2312 的扩展，向下兼容 GB2312、ASCII 码。

练习

1. 下面_____类是用来标识本地化消息的，包含对主要地理区域的地域化特征的封装。

 A. Locale　　B. ResourceBundle

 C. ListResourceBundle　　D. PropertyResourceBundle

2. ISO—639 语言列表中汉语的编码是_____。

 A. ch　　B. zh　　C. cn　　D. en

3. ISO—3166 国家（地区）列表中表示中国的编码是_____。

 A. CN　　B. US　　C. GB　　D. JP

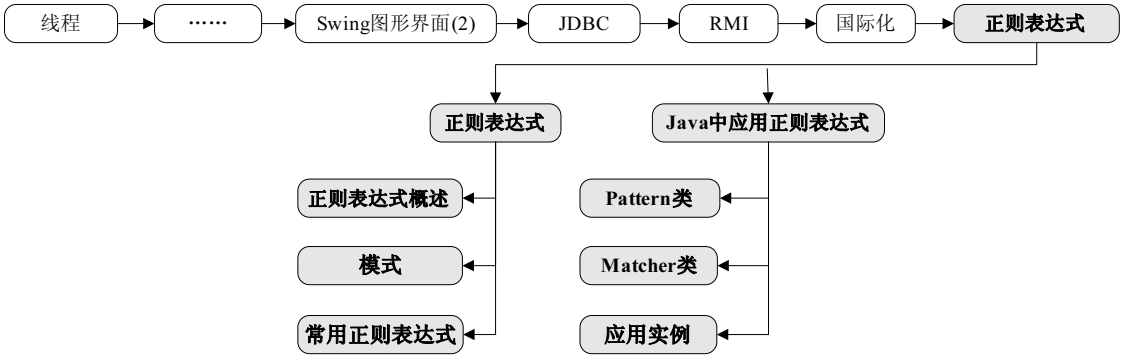
4. NumberFormat 类在_____包中。
5. 在 Java 中还提供了一个_____类，它是 DateFormat 的子类，可以更加灵活地格式化日期。
6. 将属性文件转换成对应的 Unicode 编码的命令是_____。
7. 简述 Java 资源包的命名规范。
8. 创建一个可以本地化的用户登录界面，界面中包括用户名、密码及语言选择，当选择不同语言时，界面刷新成对应的语言版本。

第 9 章　正则表达式

本章目标

- 了解正则表达式的概念
- 理解正则表达式的定义元素
- 掌握常用的正则表达式
- 理解 Java 对正则表达式的支持
- 掌握 Pattern 类的用法
- 掌握 Matcher 类的用法

学习导航



任务描述

【描述 9.D.1】

实现验证手机号码的正则表达式。

【描述 9.D.2】

实现对身份证号码（15 位、18 位）验证的正则表达式。

【描述 9.D.3】

实现对 E-mail 验证的正则表达式。

【描述 9.D.4】

演示 Pattern 类的使用。

【描述 9.D.5】

演示 Matcher 类的使用。

【描述 9.D.6】

基于 Java，使用正则表达式对用户注册信息中的手机号码、身份证号及 E-mail 地址进行初始验证。

9.1　正则表达式

在程序开发中，经常需要进行匹配、查找、替换、判断字符串等操作，而这些操作有时又比较复杂，如果用纯编码方式解决，往往会浪费程序员的时间及精力。正则表达式是一种强大而灵活的文本处理工具，在程序设计语言中有着广泛的应用，特别是用来处理字符串的匹配、查找、替换、判断等操作时。

9.1.1　正则表达式概述

正则表达式（Regular Expression）是一种描述字符串集的通用标准，它是以字符串集中各字符串的共有特征为依据的。正则表达式提供了一种紧凑的、动态的语言，能够以一种完全通用的方式来解决各种字符串处理问题。正则表达式最早出现于 Perl，目前，各操作系统及大部分程序设计语言都支持正则表达式。

正则表达式主要用于：

- **搜索与替换：**特定模式字符串的查找与替换。
- **验证：**检查某个字符串是否符合特定的匹配模式。如 E-mail 地址、HTTP 地址、FTP 地址、身份证号码等的验证。
- **解析：**从一个文本中获取有用的字符串信息，如从 HTML 文件中提取超链接。

注意　对于不同的编程语言，正则表达式的基础理论可能存在一些差异。

9.1.2　模式

模式（Pattern）是正则表达式中最重要的部分，其规定了正则表达式的匹配法则和规范。正则表达式常用的定义元素有：

- 元字符
- 字符类
- 转义字符
- 限定符
- 反义
- 逻辑或
- 分组

1. 元字符

元字符（Meta Character）是能够匹配一个位置或某个字符串的一个字符。元字符分为两类：匹配位置的元字符和匹配字符的元字符。

匹配位置的元字符有 3 个：^ 、$ 和\b，用来指示匹配开始的位置，其作用如表 9-1

所示。

表 9–1　匹配位置的元字符

字符	作用	示例
^	指示从行的开始位置开始匹配	如：^java 匹配以“java”开始的行
$	指示从行的结束位置开始匹配	如：^java$匹配以“java”开始和结尾的行
\b	匹配单词的开始或结束位置	如：\bjava 匹配以“java”开始的字符串，而“java”之前必须是单词的分界处

匹配字符的元字符有 7 个，其作用如表 9-2 所示。

表 9–2　匹配字符的元字符

字符	作用	示例
.	匹配除换行符之外的任意字符	如：^..c 匹配第 3 个字符是 c 的行，“abc”匹配成功
\w	匹配单字字符（a~z、A~Z、0~9 及下画线）	如：^..\w 匹配第 3 个字符是单字字符的行，“abc”匹配成功
\W	匹配非单字字符	如：^..\W 匹配第 3 个字符不是单字字符的行，“ab%”匹配成功
\s	匹配空白字符（如空格、制表符、换行符等）	如：^..\s 匹配第 3 个字符是空白字符的行，“ab cd”匹配成功
\S	匹配非空白字符	如：^..\S 匹配第 3 个字符是非空白字符的行，“abcd”匹配成功
\d	匹配数字字符（0~9）	如：^..\S 匹配第 3 个字符是数字字符的行，“ab3d”匹配成功
\D	匹配非数字字符	如：^..\S 匹配第 3 个字符不是数字字符的行，“abdd”匹配成功

2. 字符类

元字符只能匹配一个位置或字符集合中的一个字符，如果要匹配字符集合（如[0,1,2,3,4]）时，则需要定义匹配的字符集合。字符类就是方括号中的一个字符集，只匹配括号内的任意字符。

下述正则表达式匹配 HTML 文本中的“<H1>”、“<H2>”、“<H3>”、“<H4>”或“<H5>”。

```
<H[12345]>
```

在使用字符类进行匹配时，对于连续的字符（如：a~z、0~9 等），如果全部在括号中列举很不方便，此时可以使用范围符“-”来定义字符的范围，如上述正则表达式可以定义为：

```
<H[1-5]>
```

而对于 8 位的电话号码可以定义为：

```
\b[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]\b
```

3. 转义字符

对于诸如“^”、“$”这样的字符都有特殊意义，如果需要在正则表达式中匹配字符串中的“^”和“$”等字符时，则需要使用转义字符“\”（反斜杠）来解决这一问题。

如：

\.可以匹配字符“.”。

\\可以匹配字符“\”。

以下正则表达式匹配“www.baidu.com”。

```
www\.baidu\.com
```

常用的需要转义的字符有：.、$、^、{、[、|、]、*、+、?、\，除上述字符外，其他字符都不需要进行转义。

4. 限定符

一个元字符只能匹配一个字符，如果需要零个、一个或多个字符时，可以使用限定符。限定符用于指定允许特定字符或字符集自身重复出现的次数。有*、+、?、{n}、{n,}和{n,m}共 6 种。常用的限定符及作用如表 9-3 所示。

表 9–3　常用限定符

限定符	作用	示例
*	匹配前面的子表达式零次或多次	如：zo*能匹配“z”及“zoo”，*等价于{0,}
+	匹配前面的子表达式一次或多次	如：zo+能匹配“zo”及“zoo”，但不能匹配“z”，+等价于{1,}
?	匹配前面的子表达式零次或一次	如：do[es]?可以匹配“do”或“does”中的“doe”，?等价于{0,1}
{n}	匹配确定的 n 次，n 是一个非负整数	如：o{2}能匹配“food”中的两个“oo”
{n,}	至少匹配 n 次，n 是一个非负整数	如：o{2,}能匹配“foooood”中的所有“o”，但不能匹配“Tom”中的“o”。o{1,}等价于 o+
{n,m}	最少匹配 n 次且最多匹配 m 次，m 和 n 均为非负整数，其中 n≤m	如：o{1,3}将匹配“fooooood”中的前三个“o”。o{0,1}等价于 o?

引入限定符后对于 8 位的电话号码可以定义为：

```
\b\d{8}\b
```

下述正则表达式限制数字型字符串的整数位最多为 3 位，小数位为 2 位。

```
^\d{0,3}\.\d{2}\b
```

如果在限定符之后再添加一个字符“？”，则表示尽可能少的重复字符“？”之前的限定符号重复次数，此种匹配方式称为懒惰匹配。如果不加“？”，则默认为贪婪匹配，即匹配最长的字符串。

以下正则表达式匹配以字母 a 开头以 b 结尾的最长字符串。

```
a.*b
```

下述正则表达式匹配以字母 a 开头以 b 结尾的最短字符串，是一种懒惰匹配。

```
a.*?b
```

对于字符串“aabab”，使用正则表达式“a.*b”的匹配结果为“aabab”；使用“a.*?b”的匹配结果为“aab”和“ab”，而不会匹配“aabab”。

5. 反义

在使用正则表达式时，如果需要匹配不在字符集指定范围内的字符时，可以使用反义字符“^”（脱字符）进行声明。

下述正则表达式匹配文件名最后一个字母不为数字字符的所有的 Java 文件名。

```
.*[^0-9]\.java$
```

6. 逻辑或

在正则表达式中，可以使用字符“|”（或），该字符表示如果字符串匹配了正则表达式中字符“|”的左边或右边的规则，则该字符串匹配该正则表达式。

以下正则表达式匹配两种固定电话号码形式：一种是 4 位区位+7 位号码，另一种是 3 位区位+8 位号码。

```
0\d{2}-\d{8}|0\d{3}-\d{7}
```

如，010-88996677 和 0532-8899567 都是与该正则表达式匹配的电话号码。

注意 在使用字符“|”匹配表达式时，首先匹配字符“|”左侧部分，当左侧部分不匹配时，才匹配右侧部分。

7. 分组

可以使用“(”和“)”把一个正则表达式的全部或部分分成一个或多个组。分组之后，可以将字符“(”和字符“)”之中的表达式看成一个整体来处理。

以下正则表达式匹配“java”出现至少一次的字符串，此处将“java”看成一个整体来进行匹配。

```
(java)+
```

而下述正则表达式则可以匹配 IP（IPv4）地址。

```
(\d{1,3}\.){3}\d{1,3}
```

注意 除上述常用规则外，正则表达式还有反向引用、零宽度断言、负向零宽度断言、

递归匹配等特殊的规范，感兴趣的读者可以参考正则表达式的相关技术文档。

8. 优先级

正则表达式存在元字符、限定符等操作符或表达式。在匹配过程中，正则表达式需遵循操作符或表达式的优先级：相同优先级的从左到右进行运算，不同优先级的运算先高后低。各种操作符的优先级从高到低如表 9-4 所示。

表 9–4　优先级顺序

操作符或表达式	说明
\	转义符
(), (?:), (?=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \元字符	位置和顺序
\|	“或”操作

9.1.3　常用正则表达式

下面列举了典型的正则表达式。

1. 手机号码验证

目前国内的手机号码是以 13、18、158 或 159 开头、长度为 11 的数字型字符串，因此在验证手机号码时需要分情况进行验证。

下述代码用于实现任务描述 9.D.1，实现验证手机号码的正则表达式如下：

【描述 9.D.1】

```
\b0?(1[38]\d{9})|(15[89]\d{8})\b
```

其中：

\b 匹配单词的边界，即单词的开始或结束位置。

0?表示数字 0 可有可无，用于匹配手机号码前面加数字 0 的情况。

(1[38]\d{9})匹配以 13、18 开头的手机号码；(15[89]\d{8})匹配以 158、159 开头的手机号码。

2. 身份证号码验证

中国境内公民身份证号码为 15 位或 18 位，从左至右为：6 位地址码、6 位或 8 位出生日期、3 位顺序码和 1 位校验码（18 位身份证）。

下述代码用于实现任务描述 9.D.2，实现对 15 位身份证号码验证的正则表达式为：

【描述 9.D.2】

```
^\d{8}((0\d)|(1[0-2]))((3[01])|([0-2]\d))\d{3}$
```

其中：

(0\d)|(1[0-2])验证身份证号码中出生日期部分的 2 位月份在 01~12 范围之内。

(3[01])|([0-2]\d) 验证身份证号码中出生日期部分的 2 位日子在 01~31 范围之内。

下述代码用于实现任务描述 9.D.2，实现对 18 位身份证号码验证的正则表达式为：

```
^\d{6}((1[89])|(2\d))\d{2}((0\d)|(1[0-2]))((3[01])|([0-2]\d))\d{3}(\d|X)$
```

其中：

((1[89])|(2\d))\d{2}验证身份证号码中出生日期部分的 4 位年份，格式为 18XX、19XX 和 2XXX。

(0\d)|(1[0-2])验证身份证号码中出生日期部分的 2 位月份在 01~12 范围之内。

(3[01])|([0-2]\d) 验证身份证号码中出生日期部分的 2 位日子在 01~31 范围之内。

d{3}(\d|X)验证 3 位顺序码和 1 位校验码，其中校验码可以是数字或大写的“X”。

3. E–mail 验证

电子邮件是最常用的网络应用之一。E-mail 地址由以下几部分组成：最开始是由字母和数字组成的用户名，紧接着是一个“@”符号，后面是一个网址，而网址通常至少由两部分组成：域名及域名的后缀，它们使用“.”隔开。

下述代码用于实现任务描述 9.D.3，实现对 E-mail 验证的正则表达式为：

【描述 9.D.3】

```
\w+@\w+(\.\w+)+
```

其中：

\w+用于匹配用户名。

@匹配 E-mail 中的字符“@”。

\.匹配字符“.”。

(\.\w+)+匹配域名及域名后缀组成的字符串。

9.2 在 Java 中应用正则表达式

在 Java 中使用正则表达式需要引入 Java 的 java.util.regex 包。java.util.regex 包主要由三个类组成：Pattern、Matcher 和 PatternSyntaxException。

- Pattern 对象表示一个已编译的正则表达式。
- Matcher 是一个靠着输入的字符串来解析这个模式和完成匹配操作的对象。
- PatternSyntaxException 对象是一个未检查异常，指示正则表达式中的一个语法错误。

一个正则表达式编译成为一个 Pattern 类的对象，这个 Pattern 对象通过调用 matcher()方法来产生一个 Matcher 对象；然后通过调用该 Matcher 实例编译正则表达式，从而对目标字符串进行匹配工作。

9.2.1　Pattern 类

Pattern 对象表示一个已编译的正则表达式。Pattern 类没有提供公共的构造方法，要构建一个模式，首先必须调用公共的静态 compile()方法，它将返回一个 Pattern 对象。这个方法接受正则表达式作为第一个参数。PatternSyntaxException 对象是一个未检查异常，指示了正则表达式中的一个语法错误，需要在正则表达式解析过程中进行捕获并处理。Pattern 类的常用方法及功能见表 9-5。

表 9–5　Pattern 类的方法列表

方法名	功能说明
static Pattern compile(String regex)	将给定的表达式 regex 编译并赋值给一个 Pattern 对象
static Pattern compile(String regex , int flags)	根据指定表达式 regex 和模式标记为 flags 编译并赋值给一个 Pattern 对象
int falgs()	返回当前 Pattern 模式类的 flag 参数
public Matcher matcher(CharSequence input)	使用指定的参数 input 生成一个匹配器 Matcher 对象
public static Matcher matcher(String regex ,CharSequence input)	利用给定的正则表达式对 input 字符串进行匹配
public String pattern()	返回该 Pattern 对象所编译的正则表达式的字符串形式
public String[] split(CharSequence input)	将 input 按照 Pattern 里所包含的正则表达式进行分割
public String toString()	返回该 Pattern 的字符串形式

在使用 compile()方法构造 Pattern 对象时，可以指定模式标记（flags 参数），模式标记是可以影响正则表达式的匹配行为的标记参数，模式标记通过 Pattern 类中的静态常量定义，常用的模式标记及功能见表 9-6。

表 9–6　Pattern 类的模式标记

常量名	功能说明
Pattern.CASE_INSENSITIVE	匹配字符时与大小写无关，该标志默认只考虑 US ASCII 字符
Pattern.UNICODE_CASE	当与 CASE_INSENSITIVE 结合时，使用 Unicode 字母匹配
Pattern.MULTILINE	启用多行模式
Pattern.CANON_EQ	启用规范等价
Pattern.DOTALL	在 dotall 模式中，表达式匹配多行字符，包括行终结符，默认情况下，表达式不匹配行终结符
Pattern.UNIX_LINES	启用 UNIX 行模式

下述代码用于实现任务描述 9.D.4，演示 Pattern 类的使用。

【描述 9.D.4】 PatternDemo.java

```
public class PatternDemo {
    public static void main(String[] args) {
```

```
        String regex = ":T";
        String input = "one:two:Three:four";
        // 使用":"构造 Pattern 对象，并指定大小写无关
        Pattern p = Pattern.compile(regex,Pattern.CASE_INSENSITIVE);
        // 使用 split 方法分割字符串
        String[] items = p.split(input);
        for (String s : items) {
            System.out.println(s);
        }
        p = Pattern.compile(regex);
        // 使用 split 方法分割字符串
        items = p.split(input);
        for (String s : items) {
            System.out.println(s);
        }
    }
}
```

执行结果如下：

```
one
wo
hree:four
one:two
hree:four
```

9.2.2 Matcher 类

Matcher 类是一个依靠输入的字符串来解析这个模式和完成匹配操作的对象。与 Pattern 相似，Matcher 也没有定义公共的构造方法，需要通过调用 Pattern 对象的 matcher 方法来获得一个 Matcher 对象。Matcher 类的常用方法及功能见表 9-7。

表 9-7 Matcher 类的方法列表

方法名	功能说明
public Matcher matcher(CharSequence input)	使用指定的 input 生成一个匹配器对象
public int end()	返回查找到的子串在目标串的最后一个位置
public boolean lookingAt()	检测目标字符串是否以子串开头
public boolean find()	查找输入子串中与该模式匹配的下一个子字符串
public boolean find(int start)	从目标串的指定位置查找输入子串中与该模式匹配的下一个子字符串
public boolean matches()	判断整个字符串是否匹配，如果匹配则返回 true，否则返回 false

（续表）

方法名	功能说明
public Pattern pattern()	返回该匹配器对象对应的 Pattern 对象
public String replaceAll(String replacement)	使用给定的字符串来替换目标串中与现有 pattern 模式匹配的全部子串
public String replaceFirst(String replacement)	使用给定的字符串来替换目标串中与现有 pattern 模式匹配的第一个子串
public Matcher reset()	重新设置匹配器对象
public Matcher reset(CharSequence input)	使用新的目标串 input 重新设置匹配器对象
public int start()	返回查找到的子串在目标串的开始位置
public Matcher appendReplacement (StringBuffer sb, String replacement)	实现非结尾处的增加和替换
public StringBuffer appendTail (StringBuffer sb)	将最后一次匹配后剩余的字符串添加到一个 StringBuffer 对象中

下述代码用于实现任务描述 9.D.5，演示 Matcher 类的使用。

【描述 9.D.5】 MatcherDemo.java

```
public class MatcherDemo {
    public static void main(String[] args) {
        /*
         * 演示 start 和 end 方法的使用
         */
        String regex = "\\bTom\\b";
        String input = "Tom Tom Tom Tome";
        Pattern pattern = Pattern.compile(regex);
        // 获得一个匹配器对象
        Matcher matcher = pattern.matcher(input);
        int index = 0;
        while (matcher.find()) {
            index++;
            System.out.print(index + " Match :");
            System.out.print("start at: " + matcher.start());
            System.out.println(",end at: " + matcher.end());
        }
        /*
         * 演示 lookingAt 和 matches 方法的使用
         */
        regex = "foo";
        input = "fooooooo";
        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(input);
```

```
        System.out.println("-------------------");
        System.out.println(input + " lookingAt() " + regex + " is:"
                + matcher.lookingAt());
        System.out.println(input + " matches() " + regex + " is:"
                + matcher.matches());
        /*
         * 演示 replaceAll 和 replaceFirst 方法的使用
         */
        regex = "dog";
        input = "The dog catch a cat. All dogs.";
        String replace = "cat";
        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(input);
        String str1 = matcher.replaceAll(replace);
        String str2 = matcher.replaceFirst(replace);
        System.out.println("-------------------");
        System.out.println(str1);
        System.out.println(str2);
        /*
         * 演示 appendReplacement 和 appendTail 方法的使用
         */
        regex = "a*b";
        input = "aabfooaabfooabfoob";
        replace = "-";
        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(input);
        StringBuffer sb = new StringBuffer();
        System.out.println("-------------------");
        while (matcher.find()) {
            matcher.appendReplacement(sb, replace);
        }
        matcher.appendTail(sb);
        System.out.println(sb.toString());
    }
}
```

执行结果如下：

```
1 Match :start at: 0,end at: 3
2 Match :start at: 4,end at: 7
3 Match :start at: 8,end at: 11
-------------------
fooooooo lookingAt() foo is:true
```

```
fooooooo matches() foo is:false
-------------------
The cat catch a cat. All cats.
The cat catch a cat. All dogs.
-------------------
-foo-foo-foo-
```

matches()和 lookingAt()方法都是尝试该模式匹配输入序列，然而不同的是，matches()要求匹配整个输入字符串，而 lookingAt()是检测目标串是否以子串开头。

9.2.3　应用实例

基于 Java，使用正则表达式进行操作的一般步骤是：

01　构造一个 Pattern 对象，如：

```
Pattern pattern = Pattern.compile("[a-z]*");
```

02　构造一个 Matcher 对象，如：

```
Matcher matcher = patter.matcher(str);
```

03　根据匹配结果进行判断、替换、查找等操作，如：

```
boolean flag = matcher.matches();
```

下述代码用于实现任务描述 9.D.6，基于 Java，使用正则表达式对用户注册信息中的手机号码、身份证号及 E-mail 地址进行初始验证。

【描述 9.D.6】 RegexDemo.java

```
public class RegexDemo {
    /*
     * 验证手机号码
     */
    public static boolean verifyPhoneno(String input) {
        boolean ck = false;
        String regex = "\\b0?(1[38]\\d{9})|(15[89]\\d{8})\\b";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);
        if (matcher.matches()) {
            ck = true;
        }
        return ck;
    }
```

```
    /*
     * 验证身份证号
     */
    public static boolean verifyID(String input) {
        boolean ck = false;
        // 15 位身份证号正则表达式
        String regex15 = "^\\d{8}((0\\d)|(1[0-2]))((3[01])|([0-2]\\d))\\d{3}$";
        // 18 位身份证号正则表达式
        String regex18 = "^\\d{6}((1[89])|(2\\d))\\d{2}((0\\d)|(1[0-2]))
            ((3[01])|([0-2]\\d))\\d{3}(\\d|X)$";
        Pattern pattern = Pattern.compile(regex15);
        Matcher matcher = pattern.matcher(input);
        // 分别判断 15 位和 18 位身份证号
        if (matcher.matches()) {
            ck = true;
        } else {
            pattern = Pattern.compile(regex18);
            matcher = pattern.matcher(input);
            if (matcher.matches()) {
                ck = true;
            }
        }
        return ck;
    }
    /*
     * 验证 E-mail
     */
    public static boolean verifyEmail(String input) {
        boolean ck = false;
        String regex = "\\w+@\\w+(\\.\\w+)+";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);
        if (matcher.matches()) {
            ck = true;
        }
        return ck;
    }
    public static void main(String[] args) {
        new FrmRegist().setVisible(true);
    }
}
class FrmRegist extends JFrame implements ActionListener {
    private JPanel p;
```

```
private JLabel lblName, lblPwd, lblID, lblPhone, lblEmail;
private JTextField txtName, txtID, txtPhone, txtEmail;
private JPasswordField txtPwd;
private JButton btnOk;
public FrmRegist() {
    super("登录");
    p = new JPanel();
    p.setLayout(null);
    lblName = new JLabel("用户名");
    lblPwd = new JLabel("密　　码");
    lblID = new JLabel("身份证");
    lblPhone = new JLabel("电话");
    lblEmail = new JLabel("Email");
    txtName = new JTextField(20);
    txtPwd = new JPasswordField(20);
    txtPwd.setEchoChar('*');
    txtID = new JTextField(20);
    txtPhone = new JTextField(20);
    txtEmail = new JTextField(20);
    btnOk = new JButton("确定");
    lblName.setBounds(30, 30, 60, 25);
    txtName.setBounds(95, 30, 120, 25);
    lblPwd.setBounds(30, 60, 60, 25);
    txtPwd.setBounds(95, 60, 120, 25);
    lblID.setBounds(30, 90, 60, 25);
    txtID.setBounds(95, 90, 120, 25);
    lblPhone.setBounds(30, 120, 60, 25);
    txtPhone.setBounds(95, 120, 120, 25);
    lblEmail.setBounds(30, 150, 60, 25);
    txtEmail.setBounds(95, 150, 120, 25);
    btnOk.setBounds(60, 180, 60, 25);
    btnOk.addActionListener(this);
    p.add(lblName);
    p.add(txtName);
    p.add(lblPwd);
    p.add(txtPwd);
    p.add(lblID);
    p.add(txtID);
    p.add(lblPhone);
    p.add(txtPhone);
    p.add(lblEmail);
    p.add(txtEmail);
    p.add(btnOk);
```

```
        this.add(p);
        this.setSize(250, 250);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == btnOk) {
            String name = txtName.getText();
            String pwd = new String(txtPwd.getPassword());
            String id = txtID.getText();
            String phone = txtPhone.getText();
            String email = txtEmail.getText();
            if (name.equals("")) {
                JOptionPane.showMessageDialog(btnOk, "姓名不能为空");
                return;
            }
            if (pwd.equals("")) {
                JOptionPane.showMessageDialog(btnOk, "密码不能为空");
                return;
            }
            //验证身份证号
            if (!RegexDemo.verifyID(id)) {
                JOptionPane.showMessageDialog(btnOk, "身份证号不合法");
                return;
            }
            //验证电话号码
            if (!RegexDemo.verifyPhoneno(phone)) {
                JOptionPane.showMessageDialog(btnOk, "电话号码不合法");
                return;
            }
            //验证邮箱
            if (!RegexDemo.verifyEmail(email)) {
                JOptionPane.showMessageDialog(btnOk, "Email 不合法");
                return;
            }
            JOptionPane.showMessageDialog(btnOk, "验证通过！");
        }
    }
}
```

执行程序，界面如图 9-1 所示，输入正确的信息，将提示“验证通过”，否则提示相应的错误信息。



图 9–1　注册信息验证

对于字符串，最常用的操作就是匹配、替换和查找，java.lang.String 通过模拟 Pattern 类和 Matcher 类的部分方法，也可以支持正则表达式。表 9-8 展示了部分支持正则表达式的 String 方法及与其等价的 Pattern 类或 Matcher 类的方法。

表 9–8　String 的等价方法

String 方法名	等价方法
public boolean matches(String regex)	Pattern.matches(regex, str)
public String[] split(String regex, int limit)	Pattern.compile(regex).split(str, n)
public String replaceFirst(String regex, String replacement)	Pattern.compile(regex).matcher(str).replaceFirst(repl)
public String replaceAll(String regex, String replacement)	Pattern.compile(regex).matcher(str).replaceAll(repl)

小结

通过本章的学习，学生应该能够学会：

- 正则表达式主要用来处理字符串，如匹配字符串、查找字符串和替换字符串等。
- 一个正则表达式就是由普通字符（例如字符 a 到 z）及特殊字符（称为元字符）组成的文字模式。
- 正则表达式常用的定义元素有元字符、字符类、转义字符、限定符、反义、逻辑或、分组等。
- 正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。
- Java 提供了 Pattern 和 Matcher 类进行正则表达式匹配。
- PatternSyntaxException 用于在匹配正则表达式过程中抛出的异常。



练习

1. 在正则表达式中，_____符号用于匹配单个字符。

 A. .　　B. ?

 C. *　　D. +

2. 在正则表达式的符号中，下面_____符号优先级最高。

 A. ^　　B. *

 C. \　　D. $

3. 下面_____符号用于匹配一次或多次。

 A. _　　B. ?

 C. *　　D. +

4. 指定至少有一位数字、但不超过两个数字的正则表达式是_____。
5. 在 java.util.regex 包中有哪 3 个公共的类？描述一下它们的作用。
6. 这里是 3 个预定义的字符类：\d、\s 和\w。描述一下它们各表示什么？并使用方括号的形式将它们重写。
7. 对于\d、\s 和\w，写出两个简单的表达式，匹配它们相反的字符集。

实践篇



实践 1　线程



实践指导

实践 1.G.1

通过“生产/消费”模型演示多线程的资源竞争，以及由此产生的问题。

分析

1. 多个线程共享同一个对象，并且有一个或多个线程会修改该对象时，若不进行合理的管理，对象状态就可能产生错误的结果。
2. 在“生产/消费”模型中，资源竞争源于生产者和消费者共享同一个资源对象，故需创建三个类：资源类、生产者类和消费者类。
3. 由于需要演示资源竞争可能出现的问题，故不需要使用同步机制。

参考解决方案

1. 定义资源类 ShareValue.java，代码如下：

```
public class ShareValue {
    private int total;
    public ShareValue(int total) {
        this.total = total;
    }
    // 在 total 基础上增加传递的值
    void putValue(int value) {
        total = total + value;
    }
    // 在 total 基础上减去传递的值
    int getValue(int value) {
        if (total - value >= 0)
            total = total - value;
        else {
            value = total;
            total = 0;
            System.out.println("Empty");
        }
        return value;
    }
```

```
    // 返回 Total 现有的值
    int total() {
        return total;
    }
}
```

2. 定义消费者类 Consumer.java，代码如下：

```
public class Consumer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要减少的值
    int value;
    public Consumer(String name, ShareValue share, int value) {
        super(name);
        this.share = share;
        this.value = value;
    }
    public void run() {
        // 在减少之前，获取 sv 的 total 值
        int n = share.total();
        try {
            sleep(100);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        // 从 sv 中减去 value
        share.getValue(value);
        // 打印减少之前 total 的值 n，减去的值 value 和减少后 total 的值
        System.out.println(getName() + " , Value: " + n + "   Get  value: "
                + value + "  Total value: " + share.total());
    }
}
```

3. 定义生产者类 Producer.java，代码如下：

```
class Producer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要增加的值
    int value;
    public Producer(String name,ShareValue share, int value) {
        super(name);
        this.share = share;
```

```
        this.value = value;
    }
    public void run() {
        // 在增加之前，获取 sv 的 total 值
        int n = share.total();
        try {
            sleep(100);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        // 向 sv 中增加新值
        share.putValue(value);
        // 打印增加之前 total 的值 n，增加的值 value 和增加后 total 的值
        System.out.println(getName()+" , Value: " + n + "   Put vlaue: " + value
                + "  Total value: " + share.total());
    }
}
```

上述代码是最常见的“生产/消费”简单模型。其中 ShareValue 类代表消费的对象模型，生产者（Producer）负责向共享的 ShareValue 对象 share 中增加值（在 run()方法中调用 putValue()方法），消费者（Consumer）负责从共享的 ShareValue 对象 share 中减少值（在 run()方法中调用 getValue()方法），并且通过打印语句输出生产消费前后的数据差异。

基于此模型，执行如下测试：

```
public class PCDemo {
    public static void main(String args[]) {
        ShareValue sv = new ShareValue(0);
        Producer producer1 = new Producer("producer1",sv, 100);
        Producer producer2 = new Producer("producer2",sv, 200);
        Consumer consumer = new Consumer("consumer",sv, 50);
        producer1.start();
        consumer.start();
        producer2.start();
    }
}
```

此时程序可能的输出结果如下：

```
producer2 , Value: 0   Put vlaue: 200  Total value: 250
consumer1 , Value: 0   Get  value: 50  Total value: 250
producer1 , Value: 0   Put vlaue: 100  Total value: 250
```

从运行结果可以明显看出，无论是生产（增加操作）还是消费（减少操作），其前后操作数据都出现了错误。例如，producer2 线程对象在增加（putValue）前共享对象的值为 0，

然后向共享对象中增加了 200，而在休眠后取出的共享对象的最新值为 250，数据差异相当明显。此错误的产生就在于本例中的 Producer 线程和 Consumer 线程在执行线程任务期间进行了休眠，以模拟某些操作需要等待的过程，在此期间，我们无法确切地知道 Producer 线程何时增加一个新值，也不知道 Consumer 线程何时减少一个值，而线程的执行次序具有不确定性，即在休眠了相同的时间后，线程恢复执行的次序不一定与休眠前一致，故上述测试代码在多次运行中也可能得到不同的输出结果。

实践 1.G.2

基于上述不安全的“生产/消费”模型在资源竞争中产生的问题，使用线程同步机制实现数据和操作一致。

分析

1. 资源竞争的“不安全”性是源于对共享资源的操作不一致，不能保证在某一时刻只有一个线程对资源对象进行操作。
2. 为解决资源竞争的“不安全”性，可以使用对象锁机制。
3. Java 内部通过 synchronized 关键字实现对象锁，此关键字可用于修饰方法或代码块。
4. 针对 1.G.1 产生的问题，可以调整生产者和消费者处理的过程，使其对资源对象的操作实现同步处理。

参考解决方案

1. 调整消费者类 Consumer.java，代码如下：

```
public class Consumer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要减少的值
    int value;
    public Consumer(String name, ShareValue share, int value) {
        super(name);
        this.share = share;
        this.value = value;
    }
    public void run() {
        // 同步 share 对象，直到当前代码块运行完毕后，share 的对象锁才会释放
        synchronized (share) {
            // 在减少之前，获取 sv 的 total 值
            int n = share.total();
            try {
                sleep(100);
```

```
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            // 从 sv 中减去 value
            share.getValue(value);
            // 打印减少之前 total 的值 n，减去的值 value 和减少后 total 的值
            System.out.println(getName() + " , Value: " + n + "   Get  value: "
                    + value + "  Total value: " + share.total());
        }
    }
}
```

2. 调整生产者类 Producer.java，代码如下：

```
class Producer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要增加的值
    int value;
    public Producer(String name, ShareValue share, int value) {
        super(name);
        this.share = share;
        this.value = value;
    }
    public void run() {
        // 同步 share 对象，直到当前代码块运行完毕后，share 的对象锁才会释放
        synchronized (share) {
            // 在增加之前，获取 sv 的 total 值
            int n = share.total();
            try {
                sleep(100);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            // 向 sv 中增加新值
            share.putValue(value);
            // 打印增加之前 total 的值 n，增加的值 value 和增加后 total 的值
            System.out.println(getName() + " , Value: " + n + "   Put vlaue: "
                    + value + "  Total value: " + share.total());
        }
    }
}
```

在此基础之上，再次执行 1.G.1 的测试代码，此时程序的可能输出结果如下：

```
producer1 , Value: 0   Put vlaue: 100  Total value: 100
producer2 , Value: 100   Put vlaue: 200  Total value: 300
consumer , Value: 300   Get  value: 50  Total value: 250
```

多次运行，从结果可以明显看出，无论是生产（增加操作）还是消费（减少操作），其操作数据前后的数据都保持一致。

实践 1.G.3

假设库存值为 0 时代表没有“产品”，而库存值大于 1000 时，认为生产“过剩”，对上述“生产/消费”模型进行调整，使其符合实际应用。

分析

1. 问题描述了“生产/消费”模型中的两个状态：没有产品和过剩，对于这两个状态的一般处理是：当产品过剩时，将停止生产，并应及时通知消费者进行消费；相应地，当消费者在消费过程中发现没有足够可用于消费的资源时，应及时通知生产者进行生产。通过上述分析，此问题需要生产者和消费者能够相互通知事件的发生。
2. Java 中可以通过调用 wait()、notify()和 notifyAll()方法，使线程之间相互通知事件的发生。
3. 此处需要调整生产方法（putValue）和消费方法（getValue）。

参考解决方案

1. 调整资源类 ShareValue.java，代码如下：

```
public class ShareValue {
    private int total;
    // 用于判断对象是否为空
    private boolean empty = true;
    // 用于判断对象是否已满
    private boolean full = true;
    public ShareValue(int total) {
        this.total = total;
        if (total > 0) {
            empty = false;
        }
        if (total < 1000) {
            full = false;
        }
    }
    // 在 total 基础上增加传递的值
    synchronized void putValue(int value) {
        // 获取当前线程名
```

```
        String name = Thread.currentThread().getName();
        while (full) {
            try {
                display("Full ![ " + name + " ]waits.");
                // 进入等待状态
                this.wait();
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        total = total + value;
        if (total > 1000) {
            full = true;
        }
        display(name + ",put: " + value + ",Org value: " + (total - value)
                + ",Now value : " + total);
        // 修改对象状态
        empty = false;
        // 通知所有等待线程
        notifyAll();
    }
    // 在 total 基础上减去传递的值
    synchronized int getValue(int value) {
        String name = Thread.currentThread().getName();
        while (empty || total < value) {
            try {
                display("Empty or Not enough![ " + name + " ]waits.");
                // 进入等待状态
                this.wait();
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        display(name + " get : " + value + ",Org value : " + total
                + ",Now value : " + (total - value));
        if (total - value >= 0) {
            total = total - value;
        }
        // 修改对象状态
        if (total == 0) {
            empty = true;
        }
        if (total < 1000) {
```

```
            full = false;
        }
        // 通知所有等待线程
        notifyAll();
        return value;
    }
    // 返回 Total 现有的值
    int total() {
        return total;
    }
    // 自定义打印方法
    public void display(String str) {
        System.out.println(str);
    }
}
```

2. 调整生产者类 Producer.java，代码如下：

```
class Producer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要增加的值
    int value;
    public Producer(String name, ShareValue share, int value) {
        super(name);
        this.share = share;
        this.value = value;
    }
    public void run() {
        // 同步 sv 对象，直到当前代码块运行完毕后，sv 的对象锁才会释放
        synchronized (share) {
            try {
                sleep(100);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            // 向 sv 中增加新值
            share.putValue(value);
        }
    }
}
```

3. 调整消费者类 Consumer.java，代码如下：

```
public class Consumer extends Thread {
    // 共享的 ShareValue 对象
    ShareValue share;
    // 要减少的值
    int value;
    public Consumer(String name, ShareValue share, int value) {
        super(name);
        this.share = share;
        this.value = value;
    }
    public void run() {
        // 同步 sv 对象，直到当前代码块运行完毕后，sv 的对象锁才会释放
        synchronized (share) {
            try {
                sleep(100);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            // 从 sv 中减去 value
            share.getValue(value);
        }
    }
}
```

执行如下测试：

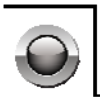
```
public class PCDemo {
    public static void main(String args[]) {
        ShareValue share = new ShareValue(0);
        Producer producer1 = new Producer("producer1", share, 100);
        Consumer consumer1 = new Consumer("consumer1", share, 300);
        Producer producer2 = new Producer("producer2", share, 950);
        Producer producer3 = new Producer("producer3", share, 50);
        producer1.start();
        consumer1.start();
        producer2.start();
        producer3.start();
    }
}
```

此时程序可能的输出结果如下：

```
producer1,put: 100,Org value: 0,Now value : 100
producer2,put: 950,Org value: 100,Now value : 1050
```

```
Full ![ producer3 ]waits.
consumer1 get : 300,Org value : 1050,Now value : 750
producer3,put: 50,Org value: 750,Now value : 800
```

“改良”后的模型运行上述测试语句，在执行 producer3 线程的时候，由于共享对象 share 的现有值为 1050，此种情况下业务逻辑认为是“过剩”，则 producer3 进入等待状态，将 share 的对象锁释放，进而 comsumer1 获取对象锁。在 comsumer1 线程执行完毕后，同时发出通知，由于 share 的现有值已变成 750，处于正常状态，所以 producer3 在获取 share 对象锁的情况下，可以继续生产，并执行通过。



知识拓展

1. 定时器

为降低多任务处理的难度，Java 中提供了两个定时器类：Timer 和 TimerTask。使用 Timer 和 TimerTask 可以很方便地实现多任务处理，使编程者从复杂的线程设计中解脱出来。

Timer 类是用来执行任务的类，它接受一个 TimerTask 作为参数。Timer 可以以 schedule 的模式执行任务，并且可以指定任务执行的频率。

TimerTask 作为 Timer 类的执行参数，定义了 Timer 类要执行的任务内容。TimerTask 事实上实现了 Runnable 接口，所以它本质上是一个线程。其定义的 run()方法是在使用 TimerTask 时必须提供实现的，类似于线程的 run()方法。TimerTask 的使用方法非常简单，只需定义其子类，并在子类中重写 run 方法即可。

Timer 类主要完成任务的管理，其常用方法及使用说明见表 1.1。

表 1.1　Timer 类的方法列表

方法名	功能说明
void cancel()	终止此计时器，丢弃所有当前已安排的任务
void schedule(TimerTask task, Date time)	到达指定的时间 time 执行指定的任务
void schedule(TimerTask task, Date time, long period)	到达指定的时间 time 开始进行重复的固定延迟 period 执行
void schedule(TimerTask task, long delay)	在指定延迟 delay 后执行指定的任务
void schedule(TimerTask task, long delay, long period)	指定的任务从指定的延迟 delay 后开始进行重复的固定延迟 period 执行
void scheduleAtFixedRate(TimerTask task, Date time, long period)	指定的任务在指定的时间 time 开始进行重复的固定延迟 period 执行
void scheduleAtFixedRate(TimerTask task, long delay, long period)	指定的任务在指定的延迟 delay 后开始进行重复的固定延迟 period 执行

下述代码使用 Timer 和 TimerTask 实现每隔 2 秒钟和 5 秒钟打印一次当前时间。

```
public class SimpleTimer extends TimerTask {
```

```
    // 任务名称
    private String name;
    public SimpleTimer(String name) {
        this.name = name;
    }
    public void run() {
        Formatter fmt = new Formatter();
        Calendar cal = Calendar.getInstance();
        // 输出年月日时分秒
        fmt.format("%tY 年%tm 月%td 日 %tA %tH 时%tM 分%tS 秒",
                cal, cal, cal, cal,cal, cal, cal);
        System.out.println(fmt);
    }
    public static void main(String[] args) {
        Timer timer = new Timer();
        // 立即执行，每 2 秒执行一次
        timer.schedule(new SimpleTimer("first "), 0, 2000);
        // 立即执行，每 5 秒执行一次
        timer.schedule(new SimpleTimer("second"), 0, 5000);
    }
}
```

执行结果如下：

```
first : 2009 年 03 月 20 日 星期五 13 时 59 分 48 秒
second: 2009 年 03 月 20 日 星期五 13 时 59 分 48 秒
first : 2009 年 03 月 20 日 星期五 13 时 59 分 50 秒
first : 2009 年 03 月 20 日 星期五 13 时 59 分 52 秒
second: 2009 年 03 月 20 日 星期五 13 时 59 分 53 秒
first : 2009 年 03 月 20 日 星期五 13 时 59 分 54 秒
```

当计划反复执行某任务时，如注重任务执行的平滑度，则使用 schedule()方法，如注重的是任务的执行频度，那么使用 scheduleAtFixedRate()方法。例如，上述代码使用了 schedule()方法，这就意味假如有一个任务因为某种原因延迟了（未按计划执行），那么余下的所有任务都要延时执行。假如我们使用 scheduleAtFixedRate()方法，这样当第一个延迟时，那么后面的就会以最快的速度紧密执行（最大限度地压缩间隔时间）。

注意 Timer 不保证任务执行地十分精确，每一个 Timer 仅对应唯一一个线程，Timer 类是线程安全的。

2. ThreadLocal

Java 在发展过程中，给我们在编写多线程应用时提供了一种新的选择：java.lang.ThreadLocal。

提出 ThreadLocal 的目的是为了方便实现多线程各自拥有同一变量的副本，而不会与其他线程副本冲突。ThreadLocal 的常用方法及使用说明见表 1.2。

表 1.2　ThreadLocal 类的方法列表

方法名	功能说明
Object get()	返回当前线程的线程局部变量副本
Object initialValue()	返回该线程局部变量的当前线程的初始值
void set(Object value)	设置当前线程的线程局部变量副本的值

其中 initialValue()方法是一个 protected 的方法，是为了子类重写提供的。该方法返回当前线程在该线程局部变量的初始值，这个方法是一个延迟调用方法，在一个线程第 1 次调用 get()或者 set()方法时才执行，且仅执行 1 次。

如果需要使用 ThreadLocal，那么需要自定义 ThreadLocal 的子类并重写该 initialValue()方法，通常使用一个内部匿名类对 ThreadLocal 进行子类化。

下述代码演示了 ThreadLocal 的使用。

```
public class ThreadLocalDemo {
    // 声明 ThreadLocal 对象
    private static ThreadLocal seqNum = new ThreadLocal() {
        // 重写 initialValue()方法
        protected synchronized Object initialValue() {
            return new Integer(0);
        }
    };
    public int get() {
        // 对本地备份值加 1
        seqNum.set(((Integer) (seqNum.get())).intValue() + 1);
        // 返回现在的值
        return ((Integer) (seqNum.get())).intValue();
    }
    public static void main(String[] args) {
        // 声明 ThreadLocal 对象 sn
        ThreadLocalDemo sn = new ThreadLocalDemo();
        // 将 sn 对象分别传递给三个线程
        TestClient t1 = new TestClient(sn);
        TestClient t2 = new TestClient(sn);
        TestClient t3 = new TestClient(sn);
        t1.start();
        t2.start();
        t3.start();
    }
    private static class TestClient extends Thread {
        private ThreadLocalDemo sn;
        public TestClient(ThreadLocalDemo sn) {
```

```
            this.sn = sn;
        }
        public void run() {
            // 打印当前线程拥有的 sn 的值
            for (int i = 0; i < 3; i++) {
                System.out.println("thread : "
                        + Thread.currentThread().getName() + "->" + sn.get());
            }
        }
    }
}
```

执行结果如下：

```
thread : Thread-2->1
thread : Thread-2->2
thread : Thread-2->3
thread : Thread-1->1
thread : Thread-1->2
thread : Thread-1->3
thread : Thread-0->1
thread : Thread-0->2
thread : Thread-0->3
```

上述代码定义了 ThreadLocal 类的对象 seqNum，并使用内部匿名类对其 initialValue()方法进行了重写，将初始值设置为 0，而在每次调用 get()方法的时候，会在现有的数值基础上加以后返回，由此实现序列操作。从运行结果可以看出，虽然将 sn 对象传递给 3 个线程，但线程是分别对自己的 sn 备份操作的。



拓展练习

练习 1.E.1

使用定时器实现每隔一秒打印当前的系统时间。

练习 1.E.2

创建一个多线程应用程序，程序中有两个子线程，一个子线程每隔一秒向集合中添加一个整数数据，另一个线程访问集合中的数据并输出。要求两个线程必须同步。

实践 2　网络编程



实践指导

实践 2.G.1

基于 Socket 模型，建立客户端和服务器的通信演示，实现客户端与服务器的双向读写操作，具体要求如下：

1. 对于客户端，既可以向服务器写数据，也可以从服务器读数据。
2. 对于服务器端，既可以向客户端写数据，也可以从客户端读数据。

分析

1. 题目要求客户端既能向服务器写数据，又能读取服务器返回的数据，所以在客户端需定义线程分别实现对客户端 Socket 写数据和读数据的功能。
2. 对于服务器端，要既可以向客户端写数据，也可以从客户端读数据，由于其处理过程也是针对 Socket 的，与客户端的 Socket 处理极其类似，所以读数据和写数据线程可以通用。

参考解决方案

1. 定义客户端（服务器端）对 Socket 读操作的线程代码如下：

```
public class ReadThread extends Thread {
    Socket socket = null;
    String clientName;
    public ReadThread(Socket socket, String clientName) {
        this.socket = socket;
        this.clientName = clientName;
    }
    public void run() {
        try {
            String line = "";
            BufferedReader is =
            new BufferedReader(new InputStreamReader(socket.getInputStream()));
            line = is.readLine();
            // 如果该字符串为 "bye"，则停止循环
            while (line != null && !line.equals("bye")) {
                System.out.println(clientName + " read :" + line);
                line = is.readLine();
```

```
            }
            // 关闭 Socket 输入流
            is.close();
            // 关闭 Socket
            socket.close();
        } catch (Exception e) {
            System.out.println("数据读取出错，无法获取 Socket 连接！ ");
            e.printStackTrace();
        }
    }
}
```

2. 定义客户端（服务器端）对 Socket 写操作的线程代码如下：

```
public class WriteThread extends Thread {
    Socket socket = null;
    String clientName;
    public WriteThread(Socket socket, String clientName) {
        this.socket = socket;
        this.clientName = clientName;
    }
    public void run() {
        try {
            PrintWriter os = new PrintWriter(socket.getOutputStream());
            BufferedReader sin = new BufferedReader(new InputStreamReader(
                    System.in));
            String readLine = sin.readLine();
            // 若读入的字符串为 "bye"则停止循环
            while (!readLine.equals("bye")) {
                os.println(readLine);
                os.flush();
                readLine = sin.readLine();
            }
            os.close();
            socket.close();
        } catch (Exception e) {
            System.out.println("数据输出错误，无法获取 Socket 连接！ ");
            e.printStackTrace();
        }
    }
}
```

3. 客户端代码如下：

```
public class Client {
    public static void main(String args[]) {
        try {
            // 向本机的 6000 端口发出客户请求
            Socket socket = new Socket("127.0.0.1", 6000);
            // 构造对 Socket 读操作的线程
            Thread read=new ReadThread(socket,"Client");
            // 构造对 Socket 写操作的线程
            Thread write=new WriteThread(socket,"Client");
            read.start();
            write.start();
        } catch (Exception e) {
            System.out.println("Error:" + e);
        }
    }
}
```

4. 服务器端代码如下：

```
public class Server {
    public static void main(String args[]) {
        ServerSocket serverSocket = null;
        Socket socket = null;
        try {
            serverSocket = new ServerSocket(6000);
            System.out.println("Server starting....");
            socket = serverSocket.accept();
            System.out.println("Accept a connection...");
            // 构造对 Socket 读操作的线程
            Thread read = new ReadThread(socket, "Server");
            // 构造对 Socket 写操作的线程
            Thread write = new WriteThread(socket, "Server");
            read.start();
            write.start();
        } catch (Exception e) {
            System.out.println("Error:"+e);
            System.exit(-1);
        }
    }
}
```

上述代码实现了客户端与服务器相互通信的功能，为区分客户端和服务器，建议在网络环境中进行代码测试。

实践 2.G.2

基于多线程模型，建立多客户端和服务器的通信演示程序，实现客户端与服务器的双向读写操作，具体要求如下：

- 对于客户端，可以实现向服务器写数据的功能，同时实现读取服务器统一下发的消息。
- 对于服务器端，能够读取所有客户端写入的数据，同时能够向在线的客户端统一下发消息。

分析

1. 题目要求客户端既能向服务器写数据，又能读取服务器返回的数据，故在客户端需定义线程分别实现对客户端 Socket 写数据和读数据的功能。
2. 对于服务器端，要能够处理多个客户端写入的数据，需定义线程处理对应客户端的 Server 端 Socket，实现 Socket 读操作。
3. 对于服务器向所有客户端统一下发消息的功能，需在服务器端通过集合维持所有客户端的本地 Socket 连接，并定义线程读取服务器消息，通过本地 Socket 集合实现消息的下发。

参考解决方案

1. 由于客户端读操作和服务器端读操作操作极其类似，可以通用，对 Socket 读操作的线程代码与实践 2.G.1 的 ReadThread.java 保持一致。
2. 此题目中，客户端的写操作只针对一个 Socket 连接，对 Socket 的写操作线程也与实践 2.G.1 的 WriteThread.java 保持一致。
3. 为区分多个客户端，在客户端运行时使用运行时参数传入客户端标识，实现代码如下：

```
public class MultiClient {
    public static void main(String args[]) throws Exception {
        if (args.length < 1) {
            throw new Exception("请输入客户端名称");
        }
        String clientName = args[0];
        try {
            Socket socket = new Socket("127.0.0.1", 4700);
            // 启动读数据线程
            Thread read = new ReadThread(socket, clientName);
            // 启动写数据线程
            Thread write = new WriteThread(socket, clientName);
            read.start();
            write.start();
        } catch (Exception e) {
```

```
            System.out.println("Error : " + e);
        }
    }
}
```

4. 服务器端消息下发需要处理多个 Socket 连接，代码如下：

```
public class ServerWriteThread extends Thread {
    // 维持各 Socket 集合
    Vector<Socket> vector = null;
    // 维持各 Socket 输出流的集合
    Vector<PrintWriter> oosVector = null;
    public ServerWriteThread(Vector<Socket> vector) {
        this.vector = vector;
        oosVector = new Vector<PrintWriter>();
    }
    public void run() {
        try {
            while (true) {
                String line;
                BufferedReader sin = new BufferedReader(new InputStreamReader(
                        System.in));
                line = sin.readLine();
                if (vector != null) {
                    // 重新构造各 Socket 集合
                    oosVector.clear();
                    for (int i = 0; i < vector.size(); i++) {
                        try {
                            Socket s = vector.elementAt(i);
                            PrintWriter os = new PrintWriter(s
                                    .getOutputStream());
                            oosVector.addElement(os);
                        } catch (Exception e) {
                            vector.removeElementAt(i);
                        }
                    }
                    // 向各客户端输出数据
                    for (int i = 0; i < oosVector.size(); i++) {
                        PrintWriter oos = oosVector.elementAt(i);
                        oos.println(line);
                        oos.flush();
                    }
                }
```

```
            }
        } catch (Exception e) {
            System.out.println("Error : " + e);
        } finally {
            // 关闭各个连接
            if (oosVector != null) {
                for (int i = 0; i < oosVector.size(); i++) {
                    PrintWriter obj = oosVector.remove(i);
                    obj.close();
                }
            }
            if (vector != null) {
                for (int i = 0; i < vector.size(); i++) {
                    Socket obj = vector.remove(i);
                    try {
                        obj.close();
                    } catch (Exception e) {
                        System.out.println("Socket 关闭出现错误： " + e);
                    }
                }
            }
        }
    }
}
```

5. 服务器端代码实现如下：

```
public class MultiServer {
    ServerSocket serverSocket = null;
    boolean listening = true;
    int clientnum = 0;
    // 使用线程安全集合类，维持所有本地 Socket 连接
    Vector<Socket> vector = new Vector<Socket>();
    public MultiServer() {
        try {
            serverSocket = new ServerSocket(4700);
            System.out.println("Server starting....");
            // 启动消息下发线程
            ServerWriteThread write = new ServerWriteThread(vector);
            write.start();
            // 永久监听客户端连接
            while (listening) {
                Socket socket = serverSocket.accept();
```

```
                clientnum++;
                // 将新接收到的 Socket 连接加入到集合中
                vector.addElement(socket);
                System.out.println("Accept  " + clientnum + "  Client");
                String client = "Server read from client[" + clientnum + "]:";
                Thread read = new ReadThread(socket, client);
                read.start();
            }
        } catch (IOException e) {
            System.out.println("Could not lienten on port : 4700.");
            System.exit(-1);
        }
        try {
            // 关闭 ServerSocket
            serverSocket.close();
        } catch (Exception e2) {
            System.out.println(e2.toString());
        }
    }
    public static void main(String args[]) throws IOException {
        new MultiServer();
    }
}
```

上述代码实现了多客户与服务器通信的功能，并实现了服务器的消息下发功能，在测试上述代码时需要注意以下几点：

- 运行客户端时需使用运行时参数提供客户端的名称。
- 上述代码演示了多客户与服务器通信的处理思路，对于客户突然关闭、读写线程之间的相互通信等情况并未进行特殊处理，具体处理过程读者可参考相关资料尝试处理，或参考后续章节的聊天室程序。
- 为区分多个客户端和服务器，建议在网络环境中进行代码测试。

实践 2.G.3

基于实践 2.G.2，完善多客户端和服务器的通信功能，实现基本的聊天功能，即当客户端向服务器端发送数据时，服务器能向所有客户端转发该客户端传入的数据。

分析

1. 基于 2.G.2 实现基本的聊天功能，客户端程序不变，只需修改服务器端程序。
2. 题目要求当客户端发送数据时，服务器向所有客户转发该客户输入的信息，即当服务器

接收到数据时，应启动服务器端信息发送线程将客户聊天信息发送给所有客户。

参考解决方案

1. 客户端应用程序与 2.G.2 保持一致，即 MultiClient.java 程序不变。
2. 服务器端因数据共享，因此将服务器端的读线程和发送线程设计成内部类。服务器端的实现代码如下：

```
public class MultiTalkServer {
    ServerSocket serverSocket = null;
    boolean listening = true;
    int clientnum = 0;
    // 使用线程安全集合类，维持所有本地 Socket 连接
    Vector<Socket> vector = new Vector<Socket>();
    public MultiTalkServer() {
        try {
            serverSocket = new ServerSocket(4700);
            System.out.println("Server starting....");
            // 永久监听客户端连接
            while (listening) {
                Socket socket = serverSocket.accept();
                clientnum++;
                // 将新接收到的 Socket 连接加入到集合中
                vector.addElement(socket);
                System.out.println("Accept  " + clientnum + "  Client");
                String client = "Server read from client[" + clientnum + "]:";
                Thread read = new ServerReadThread(socket, client);
                read.start();
            }
        } catch (IOException e) {
            System.out.println("Could not lienten on port : 4700.");
            System.exit(-1);
        }
        try {
            // 关闭 ServerSocket
            serverSocket.close();
        } catch (Exception e2) {
            System.out.println(e2.toString());
        }
    }
    public static void main(String args[]) throws IOException {
        new MultiTalkServer();
    }
```

```
// 读取客户聊天信息的线程
class ServerReadThread extends Thread {
    Socket socket = null;
    String clientName;
    public ServerReadThread(Socket socket, String clientName) {
        this.socket = socket;
        this.clientName = clientName;
    }
    public void run() {
        try {
            String line = "";
            BufferedReader is = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));
            line = is.readLine();
            // 如果该字符串为 "bye"，则停止循环
            while (line != null && !line.equals("bye")) {
                System.out.println(clientName + " read :" + line);
                // 启动消息发送线程，将客户聊天信息发送到所用客户端
                (new ServerSendThread(line)).start();
                line = is.readLine();
            }
        } catch (Exception e) {
            System.out.println("数据读取出错，无法获取 Socket 连接！ ");
            e.printStackTrace();
        }
    }
}
// 给所有客户发送聊天信息的线程
class ServerSendThread extends Thread {
    // 维持各 Socket 输出流的集合
    Vector<PrintWriter> oosVector = null;
    // 要发送的聊天信息
    String strTal = null;
    public ServerSendThread(String strTal) {
        oosVector = new Vector<PrintWriter>();
        this.strTal = strTal;
    }
    public void run() {
        try {
            if (vector != null) {
                // 重新构造各 Socket 集合
                oosVector.clear();
                for (int i = 0; i < vector.size(); i++) {
```

```
                        try {
                            Socket s = vector.elementAt(i);
                            PrintWriter os = new PrintWriter(s
                                    .getOutputStream());
                            oosVector.addElement(os);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                    // 向各客户端输出数据
                    for (int i = 0; i < oosVector.size(); i++) {
                        PrintWriter oos = oosVector.elementAt(i);
                        oos.println(strTal);
                        oos.flush();
                    }
                }
            } catch (Exception e) {
                System.out.println("Error : " + e);
            }
        }
    }
}
```

上述代码实现了多人聊天的基本功能，服务器对聊天信息进行转发。在测试上述代码时需要注意以下几点：

- 先运行服务器端，再运行客户端。
- 运行客户端时需使用运行时参数提供客户端的名称。
- 为区分多个客户端和服务器，建议在网络环境中进行代码测试。



知识拓展

1. 基于 UDP 的网络编程

UDP 是 User Datagram Protocol 的简称，是一种无连接的协议，每个数据报（Datagram）都是一个独立的信息，是一种在网络中传播的、独立的、自身包含地址信息的消息，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间及内容的正确性都是不能被保证的。它的通信双方是不需要建立连接的，对于一些不需要很高质量的应用程序来说，UDP 是一个非常好的选择，比如视频会议系统，并不要求音频和视频数据绝对的正确，只要保证连贯性就可以了，这种情况下显然使用 UDP 会更合理一些。

UDP 与 TCP 协议的区别如下：

- 使用 UDP 时，每个数据报中都给出了完整的地址信息，因此不必建立发送方和接收方的连接。对于 TCP 协议，由于它是一个面向连接的协议，在 socket 之间进行数据传输之前必然要建立连接，所以在 TCP 中必须花费时间建立连接。
- 使用 UDP 传输数据时是有大小限制的，每个被传输的数据报必须限定在 64KB 之内。而 TCP 没有这方面的限制，一旦连接建立起来，双方的 socket 就可以按统一的格式传输大量的数据了。

UDP 是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。而 TCP 是一个可靠的协议，它确保接收方完全正确地获取发送方所发送的全部数据。对数据内容正确性的检验必然占用计算机的处理时间和网络的带宽，因此 TCP 传输的效率不如 UDP 高。

java.net 包中提供了两个类 DatagramSocket 和 DatagramPacket 来支持数据报通信。DatagramSocket 用于在程序之间建立传送数据报的通信连接，DatagramPacket 则用来表示一个数据报。DatagramSocket 的常用方法及功能见表 2.1。

表 2.1　DatagramSocket 类的方法列表

方法名	功能说明
DatagramSocket()	用本地机上任何一个可用的端口创建一个套接字，这个端口号是由系统随机产生的
DatagramSocket(int port)	用一个指定的端口号 port 创建一个套接字
DatagramSocket(int port, InetAddress laddr)	用指定的端口号 port 和地址 laddr 创建一个套接字
public void receive(DatagramPacket p)	从网络上接收数据报并将其存储在 DatagramPacket 对象 p 中
public Void Send(DatagramPacket p)	将数据报对象 p 发送出去。
pubic void close()	关闭 Socket 套接字

注意　DatagramSocket 的构造方法都声明抛出非运行时异常 SocketException，程序中必须进行处理，或者捕获、或者声明抛弃。

用数据报方式编写 Client/Server 程序时，无论在客户端还是服务器端，首先都要建立一个 DatagramSocket 对象，用来接收或发送数据报，然后使用 DatagramPacket 类对象作为传输数据的载体。DatagramPacket 常用方法及功能见表 2.2。

表 2.2　DatagramPacket 类的方法列表

方法名	功能说明
DatagramPacket(byte buf[],int length)	构造长度为 length 的数据报，并且将数据存储到 buf[]中
DatagramPacket(byte buf[], int length, InetAddress addr, int port)	使用指定的地址 addr 和端口 port 构造长度为 length 的数据报，并且将数据存储到 buf[]中

（续表）

方法名	功能说明
public InetAddress getAddress()	如果是发送数据报，则获得数据报要发送的目标地址，但是如果是接收数据报则返回发送此数据报的源地址
public byte []getData()	以字节数组形式返回数据报的数据
public int getLength()	获得数据报中数据的字节数
pubic int getPort()	返回数据报中的目标地址的主机端口号

下述代码演示了一个基于 UDP 的 C/S 网络传输模型。

首先创建客户端，用于接收服务器传输的数据，代码如下：

```
public class UDPClient {
    public static void main(String[] args) {
        try {
            // 创建缓冲区
            byte[] buffer = new byte[1024];
            // 在 5000 端口创建数据报套接字
            DatagramSocket dsocket = new DatagramSocket(5000);
            System.out.println("客户端正在等待服务器端发送数据...");
            while (true) {
                // 创建 DatagramPacket 对象
                DatagramPacket packet =
                new DatagramPacket(buffer, buffer.length);
                // 无数据就等待，有数据则接收数据并存入 p 中
                dsocket.receive(packet);
                // 将接收的数据转换为 String 并输出
                String psx = new String(packet.getData(), 0, packet.getLength());
                System.out.println("来自主机："+
                packet.getAddress().toString()+"，端口号："+packet.getPort());
                System.out.println(psx);
                if ((psx == null) || psx.equals("end"))
                    break;
            }
            dsocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

服务器程序用于向客户端循环发送数据，代码如下：

```
public class UDPServer {
```

```
    public static void main(String[] args) {
        try {
            // 在 5000 端口创建数据报套接字
            DatagramSocket dsocket = new DatagramSocket(5001);
            // 封装系统标准输入设备
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                    System.in));
            System.out.println("服务端正在等待输入数据...");
            // 因为客户端运行于本地，故使用本机地址构造
            InetAddress address = InetAddress.getByName("localhost");
            while (true) {
                // 读入数据
                String str = reader.readLine();
                // 转换为数据报传输格式
                byte[] buffer = str.getBytes();
                // 向客户端发送封装好的数据报对象
                dsocket.send(new DatagramPacket(buffer, buffer.length, address,
                5000));
                if ((str == null) || str.equals("end"))
                    break;
            }
            dsocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

可以看出使用 UDP 和使用 TCP 在程序上还是有很大的区别的。UDP 的 Socket 编程是不提供监听功能的，也就是说通信双方更为平等，面对的接口是完全一样的。但是为了用 UDP 实现 C/S 结构，在使用 UDP 时可以使用 DatagramSocket.receive()来实现类似于监听的功能。因为 receive()是阻塞的函数，当它返回时，缓冲区里已经填满了接收到的一个数据报，并且可以从该数据报得到发送方的各种信息，因而可以根据读入的数据报来决定下一步的动作，这就达到了与网络监听相似的效果。

由于 UDP 是一种无连接协议，在测试时客户端与服务器端的启动顺序并没有限制。

2. 基于 UDP 的组播通信

DatagramSocket 只允许数据报发送一个目的地址，java.net 包中提供了一个 MulticastSocket 类，允许数据报以广播方式发送到该端口的所有客户。MulticastSocket 用在客户端，监听服务器广播来的数据。利用 MulticastSocket 实现广播通信可以使同时运行的多个客户程序能够接收到服务器发送来的相同的信息。

下述代码演示了基于 UDP 的 C/S 组播传输模型。

为了方便统一，定义常量类用于存储当前模型的常量信息，代码如下：

```
public class UDPConstaints {
    // 组播端口
    public static int PORT = 5003;
    // 组播地址
    public static String ADDRESS = "224.119.1.220";
    // 服务器线程读取的文件名
    public static String filename = "test.txt";
    // 服务器线程的组播时间间隔
    public static int PERIOD = 5000;
}
```

首先创建客户端，用于接收组播服务器传输的数据，代码如下：

```
public class MulticastClient {
    public static void main(String args[]) throws IOException {
        // 创建套接字
        MulticastSocket socket = new MulticastSocket(UDPConstaints.PORT);
        // 得到组播地址信息
        InetAddress address = InetAddress.getByName(UDPConstaints.ADDRESS);
        // 使用 joinGroup()将广播套接字绑定到地址上
        socket.joinGroup(address);
        DatagramPacket packet;
        for (int i = 0; i < 5; i++) {
            byte[] buf = new byte[256];
            // 创建接收数据报
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            // 构造 String 对象
            String received = new String(packet.getData());
            // 打印得到的字符串
            System.out.println("Line["+i+"]:" + received);
        }
        // 把广播套接字从地址上解除绑定
        socket.leaveGroup(address);
        socket.close();
    }
}
```

创建服务器端线程，用于将指定文件中的信息组播发送到客户端，代码如下：

```
public class MulticastServerThread extends Thread {
```

```
protected DatagramSocket socket = null;
protected BufferedReader in = null;
protected boolean hasMore = true;
public MulticastServerThread() throws IOException {
    // 创建数据报套接字
    socket = new DatagramSocket(5004);
    try {
        // 打开文件，构造相应的 BufferReader 对象
        in = new BufferedReader(new FileReader(UDPConstaints.filename));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
public void run() {
    while (hasMore) {
        try {
            // 创建缓冲区
            byte[] buf = new byte[256];
            // 从文件中读出字符串
            String dString = getNextLine();
            buf = dString.getBytes();
            InetAddress group =
            InetAddress.getByName(UDPConstaints.ADDRESS);
            // 构造 DatagramPacket 对象
            DatagramPacket packet =
            new DatagramPacket(buf, buf.length,group,UDPConstaints.PORT);
            // 发送数据报
            socket.send(packet);
            try{
                sleep(UDPConstaints.PERIOD);
            }catch(InterruptedException ie){
                ie.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
            // 标志变量置为 false，以结束循环
            hasMore = false;
        }
    }
    socket.close(); // 关闭数据报套接字
}
// 从文件中读数据
protected String getNextLine(){
```

```
        String returnValue = null;
        try {
            // 从文件中读一行，如果读到了文件尾
            if ((returnValue = in.readLine()) == null) {
                // 关闭输入流
                in.close();
                hasMore = false;
                returnValue = UDPConstaints.FILE_END;
            }
        } catch (IOException e) {
            returnValue = UDPConstaints.FILE_ERR;
            e.printStackTrace();
        }
        return returnValue;
    }
}
```

服务器执行代码如下：

```
public class MulticastServer {
    public static void main(String args[]) throws java.io.IOException {
        // 启动服务器线程
        new MulticastServerThread().start();
    }
}
```

服务器在每次发送完文件中的一行数据后，都将睡眠 5 秒钟后继续发送，直到读到文件结尾。

注意 要实现组播功能，必须使用组播地址。组播地址是保留的 D 类地址，从 224.0.0.0 到 239.255.255.255。



拓展练习

练习 2.E.1

基于多线程模型和 Socket，建立多客户端和服务器的通信演示，实现客户端与服务器的双向转发操作，即简单的聊天程序，功能如下：

- 对于客户端，可以向服务器写数据，并能接收到其他客户端传递到服务器的数据。
- 对于服务器，能够读取所有客户端写入的数据，同时能将读取的数据转发到其他在线的客户端。

实践 3　Swing 图形界面（1）



实践指导

实践 3.G.1

建立 C/S 聊天程序的客户端登录界面，具体要求如下所示。

1. 登录信息包括：用户名、密码。
2. 用户名使用文本框，密码使用密码栏。
3. 提供“确定”、“取消”和“注册”3 个按钮。

分析

1. 创建一个名为 FrmLogin 继承自 JFrame 的窗口类，作为客户端登录窗口。
2. 需要用到的组件有：1 个 JPanel、2 个 JLabel、1 个 JTextField、1 个 JPasswordField 和 3 个 JButton。
3. 使用空布局来布局界面组件，使界面整齐。

参考解决方案

1. 新建一个名为 chat 的 Java Project。
2. 创建 FrmLogin.java 程序，放在 com.haiersoft.chat.client 包中，代码如下所示：

```
public class FrmLogin extends JFrame{
    private JPanel p;
    private JLabel lblName, lblPwd;
    private JTextField txtName;
    private JPasswordField txtPwd;
    private JButton btnOk, btnCancle, btnRegist;
    public FrmLogin() {
        super("聊天室登录");
        p = new JPanel();
        p.setLayout(null);
        lblName = new JLabel("用户名");
        lblPwd = new JLabel("密　　码");
        txtName = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtPwd.setEchoChar('*');
        btnOk = new JButton("确定");
```

```
        btnCancle = new JButton("取消");
        btnRegist = new JButton("注册");
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 120, 25);
        lblPwd.setBounds(30, 60, 60, 25);
        txtPwd.setBounds(95, 60, 120, 25);
        btnOk.setBounds(30, 90, 60, 25);
        btnCancle.setBounds(95, 90, 60, 25);
        btnRegist.setBounds(160, 90, 60, 25);
        p.add(lblName);
        p.add(txtName);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnRegist);
        this.add(p);
        this.setSize(250, 170);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        FrmLogin frmLogin = new FrmLogin();
        frmLogin.setVisible(true);
    }
}
```

上述代码与理论篇中的任务描述 3.D.2 类似。

3. 运行程序，显示结果如图 3.1 所示。



图 3.1　登录界面

实践 3.G.2

创建客户端注册界面，具体要求如下所示：

1. 注册信息包括：用户名、昵称、密码、确认密码、性别和头像。

2. 用户名和昵称使用文本框。
3. 密码和确认密码使用密码栏。
4. 性别使用单选按钮，默认选项为“男”。
5. 头像使用带图片的单选按钮。
6. 提供“确定”、“取消”和“返回”3 个按钮。

分析

1. 创建一个名为 FrmRegist 继承自 JFrame 的窗口类，作为客户端注册窗口。
2. 需要用到的组件有：1 个 JPanel，6 个 JLabel、2 个 JTextField、2 个 JPasswordField、2 个 JRadioButton、3 个 JButton 和 1 个 JRadioButton 数组（用于头像的选择）。
3. 将头像图片放在 images 文件夹中。
4. 使用空布局来布局界面组件，使界面整齐。

参考解决方案

1. 在 chat 项目中添加一个名为 images 的文件夹，如图 3.2 所示。

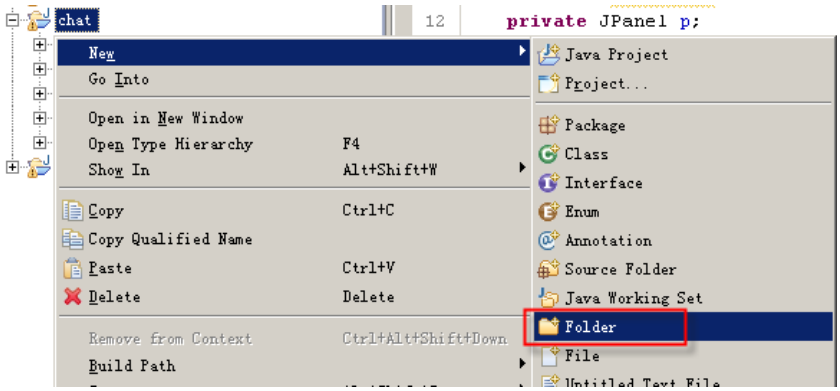


图 3.2　添加文件夹

将图片复制到此文件夹中，如图 3.3 所示。

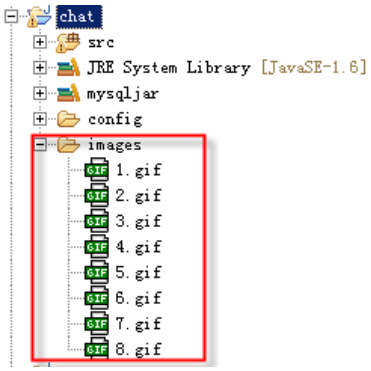


图 3.3　添加图片

2. 创建 FrmRegist.java 程序，放在 com.haiersoft.chat.client 包中，代码如下所示：

```
public class FrmRegist extends JFrame {
    private JPanel p;
    private JLabel lblName, lblNickname, lblPwd, lblRePwd, lblSex, lblImage;
    private JTextField txtName, txtNickname;
    private JPasswordField txtPwd, txtRePwd;
    private JRadioButton rbMale, rbFemale;
    private JRadioButton[] ckbImage = new JRadioButton[8];
    private JButton btnOk, btnCancle, btnReturn;
    public FrmRegist() {
        super("用户注册");
        p = new JPanel(null);
        lblName = new JLabel("用户名:");
        lblNickname = new JLabel("昵称:");
        lblPwd = new JLabel("密    码:");
        lblRePwd = new JLabel("确认密码:");
        lblSex = new JLabel("性别:");
        lblImage = new JLabel("头像:");
        txtName = new JTextField(20);
        txtNickname = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtRePwd = new JPasswordField(20);
        rbMale = new JRadioButton("男");
        rbMale.setSelected(true);
        rbFemale = new JRadioButton("女");
        JPanel pimg = new JPanel(new GridLayout(2, 5));
        ButtonGroup bgimg = new ButtonGroup();
        for (int i = 0; i < ckbImage.length; i++) {
            ckbImage[i] = new JRadioButton(String.valueOf(i + 1),
                    new ImageIcon("images/" + (i + 1) + ".gif"));
            bgimg.add(ckbImage[i]);
            pimg.add(ckbImage[i]);
        }
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        btnReturn = new JButton("返回");
        ButtonGroup bgsex = new ButtonGroup();
        bgsex.add(rbMale);
        bgsex.add(rbFemale);
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 200, 25);
        lblNickname.setBounds(30, 60, 60, 25);
```

```
        txtNickname.setBounds(95, 60, 200, 25);
        lblPwd.setBounds(30, 90, 60, 25);
        txtPwd.setBounds(95, 90, 200, 25);
        lblRePwd.setBounds(30, 120, 60, 25);
        txtRePwd.setBounds(95, 120, 200, 25);
        lblSex.setBounds(30, 150, 60, 25);
        rbMale.setBounds(95, 150, 60, 25);
        rbFemale.setBounds(160, 150, 60, 25);
        lblImage.setBounds(30, 180, 60, 25);
        pimg.setBounds(95, 180, 280, 100);
        btnOk.setBounds(90, 290, 60, 25);
        btnCancle.setBounds(155, 290, 60, 25);
        btnReturn.setBounds(220, 290, 60, 25);
        p.add(lblName);
        p.add(txtName);
        p.add(lblNickname);
        p.add(txtNickname);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(lblRePwd);
        p.add(txtRePwd);
        p.add(lblSex);
        p.add(rbMale);
        p.add(rbFemale);
        p.add(lblImage);
        p.add(pimg);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnReturn);
        this.add(p);
        this.setSize(400, 380);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        FrmRegist frmRegist = new FrmRegist();
        frmRegist.setVisible(true);
    }
}
```

在上述代码中使用循环语句创建每个带头像的 JRadioButton，并将其添加到按钮组和面

板中，代码如下：

```
for (int i = 0; i < ckbImage.length; i++) {
            ckbImage[i] = new JRadioButton(String.valueOf(i + 1),
                    new ImageIcon("images/" + (i + 1) + ".gif"));
            bgimg.add(ckbImage[i]);
            pimg.add(ckbImage[i]);
        }
```

3. 运行程序，显示结果如图 3.4 所示。



图 3.4 注册界面

实践 3.G.3

创建客户端聊天主窗口界面，具体要求如下所示：

- 界面分为 3 部分：中间显示所有聊天信息，右边显示所有在线用户，底部用于输入信息。
- 使用文本域显示所有聊天信息，文本域可以滚动但不可编辑。
- 使用文本框用于聊天信息的输入，并提供一个发送按钮。

分析

1. 创建一个名为 FrmMain 继承自 JFrame 的窗口类，作为客户端聊天主窗口。
2. 需要用到的组件有：2 个 JSplitPane（分隔面板）、1 个 JScrollPane（滚动面板）、2 个 JPanel、1 个 JTextArea、1 个 JLabel、1 个 JTextField 和 1 个 JButton。
3. 2 个 JSplitPane（分隔面板）中一个水平分隔，一个垂直分隔，将垂直分隔的面板放置在水平分隔面板的左边，如此可将界面分隔成 3 部分，如图 3.5 所示。

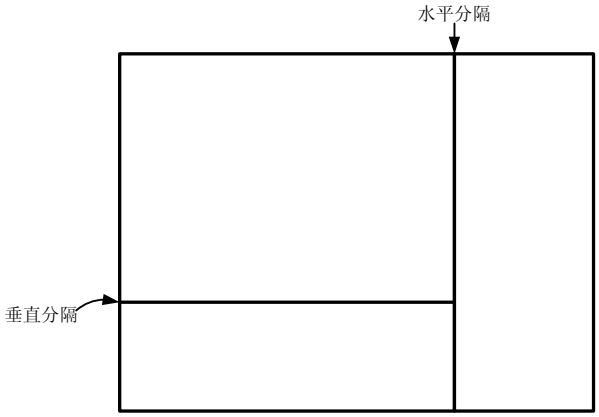


图 3.5　界面布局

注意　分隔面板（JSplitPane）和滚动面板（JScrollPane）的用法参见本章知识拓展。

参考解决方案

1. 创建 FrmMain.java 程序，放在 com.haiersoft.chat.client 包中，代码如下：

```
public class FrmMain extends JFrame {
    private JSplitPane splitPaneV, splitPaneH;
    private JScrollPane spCenter;
    private JPanel pRight;
    private JPanel pdown;
    private JTextArea txtContent;
    private JLabel lblSend;
    private JTextField txtSend;
    private JButton btnSend;
    public FrmMain() {
        super("聊天室");
        txtContent = new JTextArea();
        txtContent.setEditable(false);
        spCenter = new JScrollPane(txtContent);
        pdown = new JPanel();
        lblSend = new JLabel("输入：");
        txtSend = new JTextField(20);
        btnSend = new JButton("发送");
        pdown.add(lblSend);
        pdown.add(txtSend);
        pdown.add(btnSend);
        pRight=new JPanel(null);
        splitPaneV = new JSplitPane(JSplitPane.VERTICAL_SPLIT, spCenter, pdown);
        splitPaneV.setDividerLocation(320);
        splitPaneH = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, splitPaneV,
                pRight);
        splitPaneH.setDividerLocation(350);
```

```
        this.add(splitPaneH);
        this.setSize(500, 400);
        this.setLocation(300, 300);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        FrmMain frmMain = new FrmMain ();
        frmMain.setVisible(true);
    }
}
```

在上述代码中，设置文本域不可编辑的语句如下：

```
txtContent.setEditable(false);
```

2. 运行程序，显示结果如图 3.6 所示。

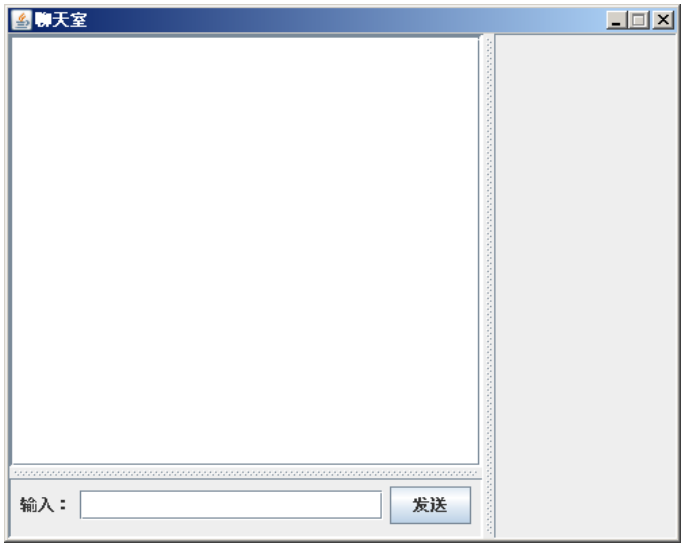


图 3.6　聊天界面

实践 3.G.4

建立 C/S 聊天程序的服务器端窗口界面的具体要求如下所示：

- 界面分为 3 部分：中间显示所有聊天信息，右边显示所有在线用户，底部用于发送系统消息。
- 使用文本域显示所有聊天信息，文本域可以滚动但不可编辑。
- 使用文本框用于系统信息的输入，并提供一个发送按钮。

分析

1. 创建一个名为 ChatServer 继承自 JFrame 的窗口类，作为服务器端窗口。
2. 需要用到的组件有：2 个 JSplitPane、2 个 JScrollPane、1 个 JPanel、1 个 JTextArea、1 个 JLabel、1 个 JTextField 和 1 个 JButton。
3. 2 个 JSplitPane 的分隔方式同 3.G.3 一致，其中一个水平分隔，一个垂直分隔。

参考解决方案

1. 创建 ChatServer.java 程序，放在 com.haiersoft.chat.server 包中，代码如下所示：

```
public class ChatServer extends JFrame {
    private JSplitPane splitPaneV, splitPaneH;
    private JScrollPane spCenter, spRight;
    private JPanel pdown;
    private JTextArea txtContent;
    private JLabel lblSend;
    private JTextField txtSend;
    private JButton btnSend;
    public ChatServer() throws Exception {
        super("服务器");
        txtContent = new JTextArea();
        txtContent.setEditable(false);
        spCenter = new JScrollPane(txtContent);
        pdown = new JPanel();
        lblSend = new JLabel("系统消息：");
        txtSend = new JTextField(20);
        btnSend = new JButton("发送");
        pdown.add(lblSend);
        pdown.add(txtSend);
        pdown.add(btnSend);
        splitPaneV = new JSplitPane(JSplitPane.VERTICAL_SPLIT, spCenter, pdown);
        splitPaneV.setDividerLocation(420);
        splitPaneH = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, splitPaneV,
                spRight);
        splitPaneH.setDividerLocation(400);
        this.add(splitPaneH);
        this.setSize(600, 500);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]) throws Exception {
        ChatServer chatServer = new ChatServer();
        chatServer.setVisible(true);
```

```
    }
}
```

2. 运行程序，显示结果如图 3.7 所示。

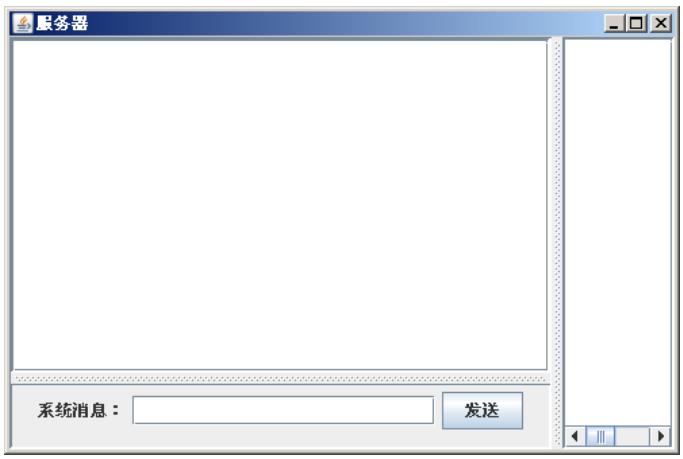


图 3.7　服务器端界面



知识拓展

1. JSplitPane

JSplitPane 是一个分隔成两部分的面板容器，可以水平或垂直分隔。其构造函数如下：

JSplitPane(int orientation,Component leftComponent,Component rightComponent)：创建一个具有指定方向、指定两部分组件的新的分隔面板。

JSplitPane 类中用于指定方向的两个静态常量如下：

- HORIZONTAL_SPLIT：水平分隔，分隔成左右两部分。
- VERTICAL_SPLIT：垂直分隔，分隔成上下部分。

JSplitPane 类常用的方法及功能见表 3.1。

表 3.1　JSplitPane 的方法列表

方法	说明
setDividerLocation(int loc)	设置分隔条的位置
setDividerLocation(double pl)	设置分隔条的位置所在的百分比
setDividerSize(int newSize)	设置分隔条的大小
setResizeWeight(double value)	指定当分隔面板改变大小时如何分配额外空间，默认值 0 表示右边/底部的组件获得所有额外空间， 1 则表示左边/顶部的组件获得所有额外空间

示例代码如下：

```
public class JSplitPaneDemo extends JFrame {
    private JSplitPane sp;
    private JPanel pLeft, pRight;
    public JSplitPaneDemo() {
        super("分隔面板");
        pLeft = new JPanel();
        pRight = new JPanel();
        // 创建一个水平分隔的分隔面板，pLeft 放在左边，pRight 放在右边
        sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, pLeft, pRight);
        // 设置分隔条的位置
        sp.setDividerLocation(200);
        // 设置分隔条的大小为 20 像素（即分隔条的粗细）
        sp.setDividerSize(20);
        // 左边组件 pLeft 获得额外空间
        sp.setResizeWeight(1);
        this.add(sp);
        this.setSize(300, 200);
        this.setLocation(300, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        JSplitPaneDemo frm = new JSplitPaneDemo();
        frm.setVisible(true);
    }
}
```

运行结果如图 3.8 所示。

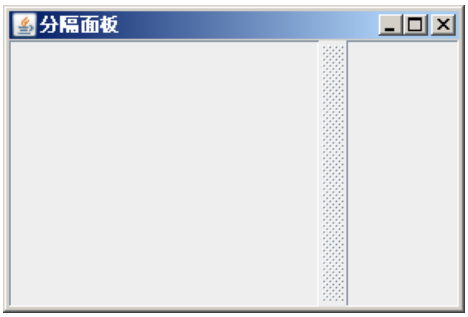


图 3.8　分隔面板

2. JScrollPane

JScrollPane 是滚动面板，可以有水平或垂直的滚动条。其常用的方法及功能见表 3.2。

表 3.2　JScrollPane 的方法列表

方法	说明
JScrollPane()	构造方法，用于创建一个空的滚动面板
JScrollPane(Component view)	构造方法，用于创建一个指定组件内容的滚动面板，当组件的内容超过视图大小就会显示水平和垂直滚动条
JScrollPane(Component view, int vsbPolicy, int hsbPolicy)	构造方法，用于创建一个指定组件内容、指定滚动条策略的滚动面板
setViewportView(Component view)	设置视图
setHorizontalScrollBarPolicy(int policy)	设置水平滚动条策略
setVerticalScrollBarPolicy(int policy)	设置垂直滚动条策略

滚动面板中的滚动条策略在 ScrollPaneConstants 接口中，该接口定义了滚动条策略静态常量值，见表 3.3。

表 3.3　ScrollPaneConstants 接口中的滚动条策略静态常量值

静态常量值	功能说明
VERTICAL_SCROLLBAR_AS_NEEDED	在需要时显示垂直滚动条
VERTICAL_SCROLLBAR_NEVER	不显示垂直滚动条
VERTICAL_SCROLLBAR_ALWAYS	总是显示垂直滚动条
HORIZONTAL_SCROLLBAR_AS_NEEDED	在需要时显示水平滚动条
HORIZONTAL_SCROLLBAR_NEVER	不显示水平滚动条
HORIZONTAL_SCROLLBAR_ALWAYS	总是显示水平滚动条

示例代码如下：

```
public class JScrollPaneDemo extends JFrame {
    private JScrollPane sp;
    private JTextArea txtContent;
    public JScrollPaneDemo() {
        super("滚动面板");
        txtContent = new JTextArea();
        // 创建一个空的滚动面板
        sp = new JScrollPane();
        // 将文本域设置到滚动面板的视图中
        sp.setViewportView(txtContent);
        // 设置滚动面板的垂直滚动条总是显示
        sp.setVerticalScrollBarPolicy(
            ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        // 设置滚动面板的水平滚动条总是显示
        sp.setHorizontalScrollBarPolicy(
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
```

```
        this.add(sp);
        this.setSize(300, 200);
        this.setLocation(300, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        JScrollPaneDemo frm = new JScrollPaneDemo();
        frm.setVisible(true);
    }
}
```

上述代码中，创建并设置滚动面板的 4 条语句可以使用下面一条语句替代：

```
sp = new JScrollPane(txtContent,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
```

运行结果如图 3.9 所示。

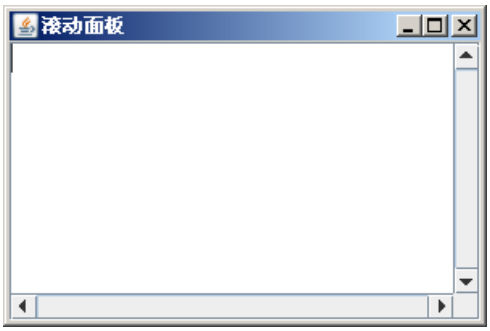
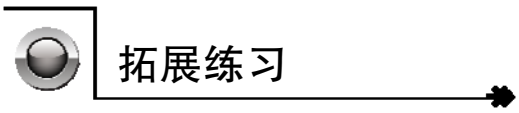


图 3.9　滚动面板



拓展练习

练习 3.E.1

实现新用户注册窗体，注册信息有用户名、密码、确认密码、性别、地址和联系电话。

练习 3.E.2

设计 8×8 的扫雷游戏界面。

实践 4　事件处理



实践指导

实践 4.G.1

在客户端登录界面中添加事件处理，具体要求如下所示：

1. 单击“确定”按钮时，先验证用户名、密码不能为空。当密码为“haiersoft”时，进入聊天室主界面，否则提示错误信息。
2. 单击“取消”按钮时，清空用户名和密码。
3. 单击“注册”按钮时，进入注册界面。

分析

1. FrmLogin 实现 ActionListener 接口。
2. 在 3 个按钮中添加行为监听对象 this。
3. 重写行为事件处理方法 actionPerformed()。在方法中，先获取事件源，对不同的事件源进行不同的处理。

参考解决方案

1. 修改 FrmLogin.java 代码，修改后的代码如下所示：

```
public class FrmLogin extends JFrame implements ActionListener {
    private JPanel p;
    private JLabel lblName, lblPwd;
    private JTextField txtName;
    private JPasswordField txtPwd;
    private JButton btnOk, btnCancle, btnRegist;
    public FrmLogin() {
        super("聊天室登录");
        p = new JPanel();
        p.setLayout(null);
        lblName = new JLabel("用户名");
        lblPwd = new JLabel("密　　码");
        txtName = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtPwd.setEchoChar('*');
```

```
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        btnRegist = new JButton("注册");
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 120, 25);
        lblPwd.setBounds(30, 60, 60, 25);
        txtPwd.setBounds(95, 60, 120, 25);
        btnOk.setBounds(30, 90, 60, 25);
        btnCancle.setBounds(95, 90, 60, 25);
        btnRegist.setBounds(160, 90, 60, 25);
        btnOk.addActionListener(this);
        btnCancle.addActionListener(this);
        btnRegist.addActionListener(this);
        p.add(lblName);
        p.add(txtName);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnRegist);
        this.add(p);
        this.setSize(250, 170);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 事件处理方法
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        // "确定"按钮
        if (source == btnOk) {
            btnOkClick();
        }
        // "取消"按钮
        if (source == btnCancle) {
            btnCancleClick();
        }
        // "注册"按钮
        if (source == btnRegist) {
            btnRegistClick();
        }
    }
```

```
    // 单击 btnOk 按钮的处理方法
    private void btnOkClick() {
        String username = txtName.getText();
        String userpwd = new String(txtPwd.getPassword());
        if (username.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "用户名不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("haiersoft")) {
            // 验证通过，隐藏登录窗口，显示聊天主窗口
            this.setVisible(false);
            FrmMain frmMain= new FrmMain();
            frmMain.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(btnOk, "错误的用户名或密码", "提示",
                    JOptionPane.WARNING_MESSAGE);
        }
    }
    // 单击 btnCancle 按钮的处理方法
    private void btnCancleClick() {
        txtName.setText("");
        txtPwd.setText("");
    }
    // 单击 btnRegist 按钮的处理方法
    private void btnRegistClick() {
        // 隐藏当前窗体
        this.setVisible(false);
        FrmRegist frmRegist = new FrmRegist();
        // 显示注册窗体
        frmRegist.setVisible(true);
    }
    public static void main(String[] args) {
        FrmLogin frmLogin = new FrmLogin();
        frmLogin.setVisible(true);
    }
}
```

2. 运行程序，显示结果如图 4.1 所示。

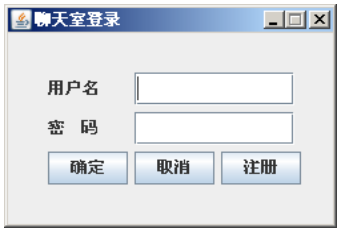


图 4.1　登录界面

单击“确定”按钮时，当条件不符合时会弹出如图 4.2 所示的对话框进行提示。



图 4.2　提示对话框

实践 4.G.2

在客户端注册界面中添加事件处理，具体要求如下所示：

1. 选中的头像图片，使用边框括起来。
2. 单击“确定”按钮时，先验证用户名、昵称、密码和确认密码不能为空，密码长度范围 6~10，确认密码和密码必须相同。当密码为“haiersoft”时，进入聊天室主界面，否则提示错误信息。
3. 单击“取消”按钮时，清空用户名、昵称、密码和确认密码，性别默认为“男”。
4. 单击“返回”按钮时，返回到登录界面。

分析

1. FrmRegist 实现 ActionListener 接口。
2. 在 3 个按钮中添加行为监听对象 this。
3. 重写行为事件处理方法 actionPerformed()。在方法中，先获取事件源，不同的事件源进行不同的处理。
4. 创建一个扩展 ItemListener 接口的监听类，监听类名为 ImageListener，用于监听头像图片。当头像图片选中时，显示边框，否则隐藏边框。

参考解决方案

1. 修改 FrmRegist.java 程序代码，修改后的代码如下所示：

```
public class FrmRegist extends JFrame implements ActionListener {
```

```
    private JPanel p;
    private JLabel lblName, lblNickname, lblPwd, lblRePwd, lblSex, lblImage;
    private JTextField txtName, txtNickname;
    private JPasswordField txtPwd, txtRePwd;
    private JRadioButton rbMale, rbFemale;
    private JRadioButton[] ckbImage = new JRadioButton[8];
    private JButton btnOk, btnCancle, btnReturn;
    // 选中的头像
    private String checkedHeadImg = null;
    public FrmRegist() {
        super("用户注册");
        p = new JPanel(null);
        lblName = new JLabel("用户名:");
        lblNickname = new JLabel("昵称:");
        lblPwd = new JLabel("密    码:");
        lblRePwd = new JLabel("确认密码:");
        lblSex = new JLabel("性别:");
        lblImage = new JLabel("头像:");
        txtName = new JTextField(20);
        txtNickname = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtRePwd = new JPasswordField(20);
        rbMale = new JRadioButton("男");
        rbMale.setSelected(true);
        rbFemale = new JRadioButton("女");
        JPanel pimg = new JPanel(new GridLayout(2, 5));
        ButtonGroup bgimg = new ButtonGroup();
        for (int i = 0; i < ckbImage.length; i++) {
            ckbImage[i] = new JRadioButton(String.valueOf(i + 1),
                    new ImageIcon("images/" + (i + 1) + ".gif"));
            ckbImage[i].addItemListener(new ImageListener());
            bgimg.add(ckbImage[i]);
            pimg.add(ckbImage[i]);
        }
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        btnReturn = new JButton("返回");
        btnOk.addActionListener(this);
        btnCancle.addActionListener(this);
        btnReturn.addActionListener(this);
        ButtonGroup bgsex = new ButtonGroup();
        bgsex.add(rbMale);
        bgsex.add(rbFemale);
        lblName.setBounds(30, 30, 60, 25);
```

```
        txtName.setBounds(95, 30, 200, 25);
        lblNickname.setBounds(30, 60, 60, 25);
        txtNickname.setBounds(95, 60, 200, 25);
        lblPwd.setBounds(30, 90, 60, 25);
        txtPwd.setBounds(95, 90, 200, 25);
        lblRePwd.setBounds(30, 120, 60, 25);
        txtRePwd.setBounds(95, 120, 200, 25);
        lblSex.setBounds(30, 150, 60, 25);
        rbMale.setBounds(95, 150, 60, 25);
        rbFemale.setBounds(160, 150, 60, 25);
        lblImage.setBounds(30, 180, 60, 25);
        pimg.setBounds(95, 180, 280, 100);
        btnOk.setBounds(90, 290, 60, 25);
        btnCancle.setBounds(155, 290, 60, 25);
        btnReturn.setBounds(220, 290, 60, 25);
        p.add(lblName);
        p.add(txtName);
        p.add(lblNickname);
        p.add(txtNickname);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(lblRePwd);
        p.add(txtRePwd);
        p.add(lblSex);
        p.add(rbMale);
        p.add(rbFemale);
        p.add(lblImage);
        p.add(pimg);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnReturn);
        this.add(p);
        this.setSize(400, 380);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 事件处理方法
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        // "确定"按钮
        if (source == btnOk) {
            btnOkClick();
```

```
        }
        // "取消"按钮
        if (source == btnCancle) {
            btnCancleClick();
        }
        // "返回"按钮
        if (source == btnReturn) {
            btnReturnClick();
        }
    }
    // 单击 btnOk 按钮的处理方法
    private void btnOkClick() {
        String username = txtName.getText();
        String nickname = txtNickname.getText();
        String userpwd = new String(txtPwd.getPassword());
        String repwd = new String(txtRePwd.getPassword());
        int sex = 0;
        if (rbFemale.isSelected()) {
            sex = 1;
        }
        if (username.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "用户名不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (nickname.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "昵称不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.length() < 6 || userpwd.length() > 10) {
        JOptionPane.showMessageDialog(btnOk, "密码长度范围在 6~10 之间", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (repwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "确认密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
```

```
        }
        if (!repwd.equals(userpwd)) {
            JOptionPane.showMessageDialog(btnOk, "确认密码和密码必须相同", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("haiersoft")) {
            JOptionPane.showMessageDialog(btnOk, "注册成功！", "提示",
                    JOptionPane.INFORMATION_MESSAGE);
            // 调用 btnCancleClick()方法，清空信息
            btnCancleClick();
        } else {
            JOptionPane.showMessageDialog(btnOk, "注册失败！", "提示",
                    JOptionPane.INFORMATION_MESSAGE);
        }
    }
    // 单击 btnCancle 按钮的处理方法
    private void btnCancleClick() {
        // 清空信息
        txtName.setText("");
        txtNickname.setText("");
        txtPwd.setText("");
        txtRePwd.setText("");
        rbMale.setSelected(true);
    }
    // 单击 btnReturn 按钮的处理方法
    private void btnReturnClick() {
        // 隐藏当前窗体
        this.setVisible(false);
        FrmLogin frmLogin = new FrmLogin();
        // 显示登录窗体
        frmLogin.setVisible(true);
    }
    // 头像选择监听类
    class ImageListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            JRadioButton source = (JRadioButton) e.getSource();
            if (source.isSelected()) {
                // 显示边框
                source.setBorderPainted(true);
                checkedHeadImg = source.getText() + ".gif";
                System.out.println(checkedHeadImg);
            } else {
                // 隐藏边框
```

```
                source.setBorderPainted(false);
            }
        }
    }
    public static void main(String[] args) {
        FrmRegist frm = new FrmRegist ();
        frm.setVisible(true);
    }
}
```

2. 运行程序，单击头像图片 3，显示结果如图 4.3 所示。



图 4.3　显示结果

单击“确定”按钮时，当条件不符合时会弹出如图 4.4 所示的对话框进行提示。



图 4.4　提示对话框

实践 4.G.3

优化 4.G.1 和 4.G.2，具体要求如下所示：

1. 窗口对象创建之后都保存到一个 Hashtable 容器中。
2. 当窗口之间进行切换时，从容器中获取该窗口对象，并显示。

3. 将 FrmRegist.java 中的 main()方法去掉，使聊天室客户端的入口界面是登录界面。

分析

1. 使用 Hashtable 容器存放窗口对象，当某个窗口需要显示时先从容器中提取此窗口对象，如果为 null，则说明容器中无此窗口对象，创建此窗口对象并保存到容器中；如果不为 null，则说明容器中有此窗口对象，直接显示即可。此种方式避免窗口对象的不断创建，造成内存空间的浪费，从而优化系统性能。
2. 创建一个名为 RongQi 的类，该类中定义一个静态的 Hashtable<JFrame>集合对象，并提供对集合存取的两个静态方法。RongQi 类用于在程序运行期间作为容器，存放 JFrame 类型的窗口对象。
3. 修改 FrmLogin.java 和 FrmRegist.java 代码，切换窗口时使用 RongQi 类。
4. 去掉 FrmRegist.java 中的 main()方法，通过单击登录窗口中的“注册”按钮，才能进入注册窗口。确保聊天室客户端只有一个入口。

参考解决方案

1. 创建 RongQi.java 程序，放在 com.haiersoft.util 包中，代码如下所示：

```
public class RongQi{
private static Hashtable<String,JFrame> table=new Hashtable<String,JFrame>();
    public static JFrame getFrm(String name){
        return table.get(name);
    }
    public static void putFrm(String name,JFrame frm){
        table.put(name,frm);
    }
}
```

2. 修改 FrmLogin.java 代码，修改后代码如下所示：

```
public class FrmLogin extends JFrame implements ActionListener {
    private JPanel p;
    private JLabel lblName, lblPwd;
    private JTextField txtName;
    private JPasswordField txtPwd;
    private JButton btnOk, btnCancle, btnRegist;
    public FrmLogin() {
        super("聊天室登录");
        p = new JPanel();
        p.setLayout(null);
        lblName = new JLabel("用户名");
        lblPwd = new JLabel("密    码");
```

```
        txtName = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtPwd.setEchoChar('*');
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        btnRegist = new JButton("注册");
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 120, 25);
        lblPwd.setBounds(30, 60, 60, 25);
        txtPwd.setBounds(95, 60, 120, 25);
        btnOk.setBounds(30, 90, 60, 25);
        btnCancle.setBounds(95, 90, 60, 25);
        btnRegist.setBounds(160, 90, 60, 25);
        btnOk.addActionListener(this);
        btnCancle.addActionListener(this);
        btnRegist.addActionListener(this);
        p.add(lblName);
        p.add(txtName);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnRegist);
        this.add(p);
        this.setSize(250, 170);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 事件处理方法
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        // "确定"按钮
        if (source == btnOk) {
            btnOkClick();
        }
        // "取消"按钮
        if (source == btnCancle) {
            btnCancleClick();
        }
        // "注册"按钮
        if (source == btnRegist) {
```

```
            btnRegistClick();
        }
    }
    // 单击 btnOk 按钮的处理方法
    private void btnOkClick() {
        String username = txtName.getText();
        String userpwd = new String(txtPwd.getPassword());
        if (username.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "用户名不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("haiersoft")) {
            // 验证通过，隐藏登录窗口，显示聊天主窗口
            this.setVisible(false);
            JFrame frmMain = RongQi.getFrm("FrmMain");
            if (frmMain == null) {
                frmMain = new FrmMain();
                RongQi.putFrm("FrmMain", frmMain);
            }
            frmMain.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(btnOk, "错误的用户名或密码", "提示",
                    JOptionPane.WARNING_MESSAGE);
        }
    }
    // 单击 btnCancle 按钮的处理方法
    private void btnCancleClick() {
        txtName.setText("");
        txtPwd.setText("");
    }
    // 单击 btnRegist 按钮的处理方法
    private void btnRegistClick() {
        // 隐藏当前窗体
        this.setVisible(false);
        // 从容器中获取注册窗体
        JFrame frmRegist = RongQi.getFrm("FrmRegist");
        // 如果为 null，则创建注册窗体，并放入容器中
```

```
        if (frmRegist == null) {
            frmRegist = new FrmRegist();
            RongQi.putFrm("FrmRegist", frmRegist);
        }
        // 显示注册窗体
        frmRegist.setVisible(true);
    }
    public static void main(String[] args) {
        FrmLogin frmLogin = new FrmLogin();
        frmLogin.setVisible(true);
        RongQi.putFrm("FrmLogin", frmLogin);
    }
}
```

3. 修改 FrmRegist.java 程序代码，修改后的代码如下所示：

```
public class FrmRegist extends JFrame implements ActionListener {
    private JPanel p;
    private JLabel lblName, lblNickname, lblPwd, lblRePwd, lblSex, lblImage;
    private JTextField txtName, txtNickname;
    private JPasswordField txtPwd, txtRePwd;
    private JRadioButton rbMale, rbFemale;
    private JRadioButton[] ckbImage = new JRadioButton[8];
    private JButton btnOk, btnCancle, btnReturn;
    // 选中的头像
    private String checkedHeadImg = null;
    public FrmRegist() {
        super("用户注册");
        p = new JPanel(null);
        lblName = new JLabel("用户名:");
        lblNickname = new JLabel("昵称:");
        lblPwd = new JLabel("密    码:");
        lblRePwd = new JLabel("确认密码:");
        lblSex = new JLabel("性别:");
        lblImage = new JLabel("头像:");
        txtName = new JTextField(20);
        txtNickname = new JTextField(20);
        txtPwd = new JPasswordField(20);
        txtRePwd = new JPasswordField(20);
        rbMale = new JRadioButton("男");
        rbMale.setSelected(true);
        rbFemale = new JRadioButton("女");
        JPanel pimg = new JPanel(new GridLayout(2, 5));
```

```
        ButtonGroup bgimg = new ButtonGroup();
        for (int i = 0; i < ckbImage.length; i++) {
            ckbImage[i] = new JRadioButton(String.valueOf(i + 1),
                    new ImageIcon("images/" + (i + 1) + ".gif"));
            ckbImage[i].addItemListener(new ImageListener());
            bgimg.add(ckbImage[i]);
            pimg.add(ckbImage[i]);
        }
        btnOk = new JButton("确定");
        btnCancle = new JButton("取消");
        btnReturn = new JButton("返回");
        btnOk.addActionListener(this);
        btnCancle.addActionListener(this);
        btnReturn.addActionListener(this);
        ButtonGroup bgsex = new ButtonGroup();
        bgsex.add(rbMale);
        bgsex.add(rbFemale);
        lblName.setBounds(30, 30, 60, 25);
        txtName.setBounds(95, 30, 200, 25);
        lblNickname.setBounds(30, 60, 60, 25);
        txtNickname.setBounds(95, 60, 200, 25);
        lblPwd.setBounds(30, 90, 60, 25);
        txtPwd.setBounds(95, 90, 200, 25);
        lblRePwd.setBounds(30, 120, 60, 25);
        txtRePwd.setBounds(95, 120, 200, 25);
        lblSex.setBounds(30, 150, 60, 25);
        rbMale.setBounds(95, 150, 60, 25);
        rbFemale.setBounds(160, 150, 60, 25);
        lblImage.setBounds(30, 180, 60, 25);
        pimg.setBounds(95, 180, 280, 100);
        btnOk.setBounds(90, 290, 60, 25);
        btnCancle.setBounds(155, 290, 60, 25);
        btnReturn.setBounds(220, 290, 60, 25);
        p.add(lblName);
        p.add(txtName);
        p.add(lblNickname);
        p.add(txtNickname);
        p.add(lblPwd);
        p.add(txtPwd);
        p.add(lblRePwd);
        p.add(txtRePwd);
        p.add(lblSex);
```

```
        p.add(rbMale);
        p.add(rbFemale);
        p.add(lblImage);
        p.add(pimg);
        p.add(btnOk);
        p.add(btnCancle);
        p.add(btnReturn);
        this.add(p);
        this.setSize(400, 380);
        this.setLocation(300, 300);
        // 设置窗体不可改变大小
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    // 事件处理方法
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        // 确定按钮
        if (source == btnOk) {
            btnOkClick();
        }
        // 取消按钮
        if (source == btnCancle) {
            btnCancleClick();
        }
        // 返回按钮
        if (source == btnReturn) {
            btnReturnClick();
        }
    }
    // 单击 btnOk 按钮的处理方法
    private void btnOkClick() {
        String username = txtName.getText();
        String nickname = txtNickname.getText();
        String userpwd = new String(txtPwd.getPassword());
        String repwd = new String(txtRePwd.getPassword());
        int sex = 0;
        if (rbFemale.isSelected()) {
            sex = 1;
        }
        if (username.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "用户名不能为空", "提示",
```

```
                JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (nickname.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "昵称不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.length() < 6 || userpwd.length() > 10) {
        JOptionPane.showMessageDialog(btnOk, "密码长度范围在 6~10 之间", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (repwd.equals("")) {
            JOptionPane.showMessageDialog(btnOk, "确认密码不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (!repwd.equals(userpwd)) {
            JOptionPane.showMessageDialog(btnOk, "确认密码和密码必须相同", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (userpwd.equals("haiersoft")) {
            JOptionPane.showMessageDialog(btnOk, "注册成功！", "提示",
                    JOptionPane.INFORMATION_MESSAGE);
            // 调用 btnCancleClick()方法，清空信息
            btnCancleClick();
        } else {
            JOptionPane.showMessageDialog(btnOk, "注册失败！", "提示",
                    JOptionPane.INFORMATION_MESSAGE);
        }
    }
    // 单击 btnCancle 按钮的处理方法
    private void btnCancleClick() {
        // 清空信息
        txtName.setText("");
```

```
        txtNickname.setText("");
        txtPwd.setText("");
        txtRePwd.setText("");
        rbMale.setSelected(true);
    }
    // 单击 btnReturn 按钮的处理方法
    private void btnReturnClick() {
        // 隐藏当前窗体
        this.setVisible(false);
        JFrame frmLogin = RongQi.getFrm("FrmLogin");
        if (frmLogin == null) {
            frmLogin = new FrmLogin();
            RongQi.putFrm("FrmLogin", frmLogin);
        }
        // 显示登录窗体
        frmLogin.setVisible(true);
    }
    // 头像选择监听类
    class ImageListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            JRadioButton source = (JRadioButton) e.getSource();
            if (source.isSelected()) {
                // 显示边框
                source.setBorderPainted(true);
                checkedHeadImg = source.getText() + ".gif";
                System.out.println(checkedHeadImg);
            } else {
                // 隐藏边框
                source.setBorderPainted(false);
            }
        }
    }
}
```



知识拓展

1. AdjustmentListener

AdjustmentListener 是一个用于接收调整事件的监听器接口，用于滚动条、滑块等可以调整的组件，该接口中用于处理调整事件的方法如下：

void adjustmentValueChanged(AdjustmentEvent e)：当可调整的值发生更改时调用该方法。

使用 addAdjustmentListener()方法可以将调整监听对象添加到组件中。

调整事件示例如下：

```
public class AdjustmentDemo extends JFrame implements AdjustmentListener {
    JPanel p;
    JLabel lblRed, lblGreen, lblBlue;
    JScrollBar sbRed, sbGreen, sbBlue;
    public AdjustmentDemo() {
        super("调色板");
        p = new JPanel();
        lblRed = new JLabel("红色");
        lblGreen = new JLabel("绿色");
        lblBlue = new JLabel("蓝色");
        // 创建一个水平的、初始位置为 0、跨度为 0、最小值为 0、最大值为 255 的滚动条
        sbRed = new JScrollBar(JScrollBar.HORIZONTAL, 0, 0, 0, 255);
        sbGreen = new JScrollBar(JScrollBar.HORIZONTAL, 0, 0, 0, 255);
        sbBlue = new JScrollBar(JScrollBar.HORIZONTAL, 0, 0, 0, 255);
        sbRed.addAdjustmentListener(this);
        sbGreen.addAdjustmentListener(this);
        sbBlue.addAdjustmentListener(this);
        p.add(lblRed);
        p.add(sbRed);
        p.add(lblGreen);
        p.add(sbGreen);
        p.add(lblBlue);
        p.add(sbBlue);
        this.add(p);
        this.setSize(300, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void adjustmentValueChanged(AdjustmentEvent e) {
        int r = sbRed.getValue();
        int g = sbGreen.getValue();
        int b = sbBlue.getValue();
        // 创建指定参数的颜色对象
        Color c = new Color(r, g, b);
        p.setBackground(c);
    }
    public static void main(String args[]) {
        new AdjustmentDemo().setVisible(true);
    }
}
```

运行结果如图 4.5 所示。

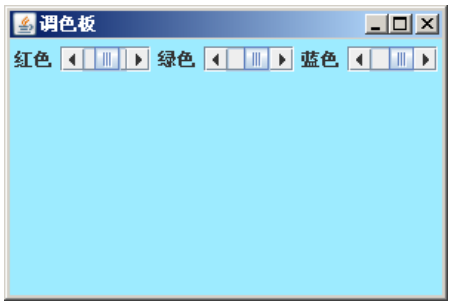


图 4.5　运行结果

2. 人物眼球转动

下面是使用线程让人物眼球转动的示例：

```
public class EyeRunDemo extends JFrame implements Runnable {
    JPanel p;
    JLabel labelAnimation;
    ImageIcon picture[];
    Thread t1;
    public EyeRunDemo() {
        super("人物眼球转动");
        p = new JPanel();
        labelAnimation = new JLabel("");
        p.add(labelAnimation);
        picture = new ImageIcon[8];
        for (int i = 0; i < picture.length; i++)
            picture[i] = new ImageIcon("images/Freedom" + (i + 1) + ".gif");
        this.add(p);
        this.setSize(400, 300);
        this.setVisible(true);
        t1 = new Thread(this);
        t1.start();
    }
    public void run() {
        while (true) {
            for (int i = 0; i < picture.length; i++) {
                labelAnimation.setIcon(picture[i]);
                try {
                    t1.sleep(100);
                } catch (InterruptedException e) {
                    System.out.println("Thread interrupted");
                }
            }
```

```
        }
    }
    public static void main(String args[]) {
        new EyeRunDemo();
    }
}
```

运行结果如图 4.6 所示。



图 4.6　运行结果

拓展练习

练习 4.E.1

实现齐鲁风采彩票系统（30 选 7、23 选 5、双色球和 3D）。

练习 4.E.2

实现 8×8 的扫雷游戏程序。

实践 5　Swing 图形界面（2）



实践指导

实践 5.G.1

实现客户端聊天主窗口中的发送和接收聊天信息的功能，具体要求如下所示：

- 建立与服务器连接的套接字，用于传递聊天信息。服务器的 IP 地址和端口信息从文件中获取。
- 发送聊天信息时，信息不能为空。
- 创建一个接收服务器发送信息的线程，将接收的信息显示在文本域中。

分析

1. 创建一个 Config.ini 文件，存储服务器的 IP 地址和端口信息。
2. 创建一个名为 Config 的类，用于读取文件中指定键对应的值信息。
3. 修改 FrmMain.java 程序代码，增加与服务器连接的功能。
4. 在“发送”按钮上添加监听，实现发送聊天信息的功能。
5. 创建一个接收服务器发送信息的线程，在构造方法中创建并启动此线程。

参考解决方案

1. 创建 Config.ini 文件，放在 config 文件夹下，如图 5.1 所示。

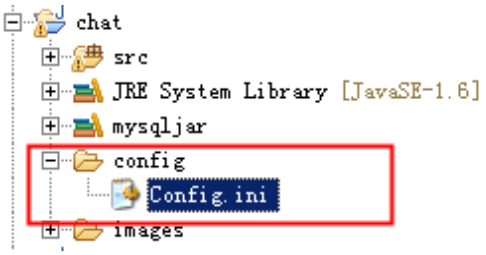


图 5.1　创建 Config.ini 文件

Config.ini 文件内容如下所示：

```
server=localhost
msgport=8000
userport=8001
```

注意　将服务器的 IP 地址和端口放在文件中，便于在实际运行时，如服务器地址改变，不必更改程序代码，只需修改文件中的值即可。例如，聊天室服务器端应用程序运行在 192.168.0.1 机器上（非本机 localhost），则应将 server 的值修改为 192.168.0.1，即“server=192.168.0.1”。注意按照“键=值”形式存放信息，每行只存放一个“键=值”并需要回车。

2. 创建 Config.java 程序，放在 com.haiersoft.util 包中，代码如下所示：

```
//配置类
public class Config {
    private static Properties p = null;
    static {
        try {
            p = new Properties();
            // 加载配置文件
            p.load(new FileInputStream("config/Config.ini"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 获取键对应的值
    public static String getValue(String key) {
        return p.get(key).toString();
    }
}
```

上述代码中使用静态块创建 Properties 对象，使用 Properties 对象的 load()方法加载 Config.ini 文件；并提供一个 getValue()静态方法，在此方法中通过调用 Properties 对象的 get()方法获取文件中指定键所对应的值。

3. 修改 FrmMain.java 程序代码，增加与服务器连接的功能：

在 FrmMain 类中声明用于聊天信息传递的 Socket 和输入/输出流，代码如下所示：

```
//用于消息传递的套接字
    private Socket socketMsg;
//用于消息的输入/输出流
    private PrintWriter printMsg;
    private BufferedReader readMsg;
```

在 FrmMain 类中编写连接服务器聊天信息的 createMsgConnection()方法，代码如下所示：

```
    // 建立聊天信息链接
    private void createMsgConnection() {
        try {
```

```
            socketMsg = new Socket(Config.getValue("server"), Integer
                    .parseInt(Config.getValue("msgport")));
            printMsg = new PrintWriter(socketMsg.getOutputStream(), true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

在 FrmMain 类的构造方法中调用 createMsgConnection()方法，代码如下所示：

```
createMsgConnection();
```

4. 修改 FrmMain.java 程序代码，实现发送聊天信息的功能：

FrmMain 类实现 ActionListener 接口，代码如下所示：

```
public class FrmMain extends JFrame implements ActionListener
```

在类中添加如下 3 个属性：

```
// 当前聊天用户的用户名
public static String username = "haier";
// 当前聊天用户的昵称
public static String nickname = "海尔";
// 当前聊天用户的头像
public static String headimg = "8.gif";
```

在“发送”按钮上添加监听，代码如下所示：

```
btnSend.addActionListener(this);
```

编写事件处理方法，代码如下所示：

```
public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == btnSend) {
            String str = txtSend.getText();
            if (!str.equals("")) {
                // 网络通信
                printMsg.println(nickname + ":" + str);
                printMsg.flush();
                txtSend.setText("");
            }
        }
    }
```

5. 创建接收服务器发送信息的线程，代码如下所示：

```
// 从服务器获取聊天信息的线程
class GetMsgFromServer extends Thread {
    private String strMsg = null;
    public GetMsgFromServer() {
        super();
    }
    public void run() {
        try {
            readMsg = new BufferedReader(
                new InputStreamReader(socketMsg.getInputStream()));
            do {
                strMsg = readMsg.readLine();
                // 在文本域显示聊天信息
                txtContent.append(strMsg + "\n");
            } while (strMsg!=null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

在 FrmMain 类的构造方法中创建此线程并启动，代码如下所示：

```
new GetMsgFromServer().start();
```

实践 5.G.2

实现客户端聊天主窗口中的发送和接收用户信息的功能，具体要求如下所示：

- 建立与服务器连接的套接字，用于传递用户信息。服务器的 IP 地址和端口信息从文件中获取。
- 发送的用户信息封装在可序列化类中。
- 创建一个接收服务器发送用户信息的线程，将接收的用户列表显示在右边的面板中。

分析

1. 创建一个名为 User 的可序列化类，用于存放用户信息。
2. 修改 FrmMain.java 程序代码，增加与服务器连接的功能，并将当前用户信息发送给服务器。
3. 创建一个接收服务器发送的用户列表信息的线程，在构造方法中创建并启动此线程。

参考解决方案

1. 创建一个名为 User 的可序列化类，用于存放用户信息，代码如下所示：

```
//User 可序列化类，便于在网络中传递信息
public class User implements Serializable {
    // 用户名
    public String userName = "";
    // 昵称
    public String nickName = "";
    // 头像
    public String headImg = "";
    public User(String username, String nickname, String headimg) {
        this.userName = username;
        this.nickName = nickname;
        this.headImg = headimg;
    }
```

2. 修改 FrmMain.java 程序代码，增加与服务器连接的功能：

 在 FrmMain 类中声明用于用户信息传递的 Socket 和输入/输出流，代码如下所示：

```
//用于用户传递的套接字
private Socket socketUser;
//用于用户信息的输入/输出流
private ObjectOutputStream printUser;
private BufferedReader readUser;
```

 在 FrmMain 类中编写 createUserListConnection ()方法，在此方法中建立连接服务器套接字，并将用户信息封装到可序列化类中发送给服务器。代码如下所示：

```
// 建立用户信息连接
    private void createUserListConnection() {
        try {
            socketUser = new Socket(Config.getValue("server"),
                Integer.parseInt(Config.getValue("userport")));
            printUser = new ObjectOutputStream(
                socketUser.getOutputStream());
            // 封装当前用户的信息
            User user = new User(username, nickname, headimg);
            // 将当前用户发送给服务器
            printUser.writeObject((User) user);
            printUser.flush();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }

    }
```

在 FrmMain 类的构造方法中调用 createUserListConnection ()方法，代码如下所示：

```
createUserListConnection();
```

3. 创建接收服务器发送的所有用户列表信息的线程，代码如下所示：

```
// 从服务器获取所有聊天用户信息的线程
class GetUserListFromServer extends Thread {
    public GetUserListFromServer() {
        super();
    }
    public void run() {
        try {
            readUser = new BufferedReader(
                new InputStreamReader(socketUser.getInputStream()));
            do {
                String line = readUser.readLine();
                if (line != null) {
                StringTokenizer st = new StringTokenizer(line, "&");
                pRight.removeAll();
                int y=10;
                while (st.hasMoreTokens()) {
                    String name = st.nextToken();
                    String img = st.nextToken();
                    System.out.println(name + ":" + img);
                    JLabel userNode = new JLabel(name, new ImageIcon(
                        "images/" + img), JLabel.LEFT);
                    userNode.setBounds(10, y, 140, 40);
                    pRight.add(userNode);
                    y+=45;
                }
                pRight.setSize(150,300);
                splitPaneH.setDividerLocation(350);
                FrmMain.this.setSize(0, 0);
                FrmMain.this.setSize(500, 400);
            }
        } while (true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

注意 服务器发送的用户列表信息中包括用户昵称和头像，每项信息用“&”符号进行间隔。使用 StringTokenizer 可以对带特殊符号的字符串进行解析。

在 FrmMain 类的构造方法中创建此线程并启动，代码如下所示：

```
new GetUserListFromServer().start();
```

实践 5.G.3

实现聊天室服务器端功能，具体要求如下所示：

- 接收所有用户发送的聊天信息，群发聊天信息并显示在文本域中。
- 接收所有用户发送的用户信息，更新用户列表，并群发用户列表信息。
- 在右边的面板中使用树显示用户列表信息。
- 实现群发系统消息的功能。

分析

1. 在 ChatServer 类中声明所需要的树、服务套接字及集合。其中，服务套接字有两个，分别是用于接收聊天和用户信息的套接字；集合有 8 个，其中 4 个用于存放与聊天消息有关内容：消息套接字、消息输出流、消息输入流和消息列表；另外 4 个用于存放与用户有关的内容：用户套接字、用户输出流、用户输入流和用户列表信息。
2. 创建接收客户端发送的有关用户套接字的线程。
3. 创建接收客户端发送的有关消息套接字的线程。
4. 创建接收客户发送聊天信息的线程。
5. 创建向所用客户端发送消息的线程。
6. 在 ChatServer 类的构造方法中创建线程并启动。
7. 实现单击“发送”按钮，群发系统消息的事件处理。

参考解决方案

1. 在 ChatServer 类中声明所需要的树、服务套接字及集合，代码如下所示：

```
private DefaultMutableTreeNode root;
private DefaultTreeModel model;
private JTree tree;
// 消息服务套接字
private ServerSocket msgServerSocket;
// 用户服务套接字
private ServerSocket userServerSocket;
private ArrayList<Socket> msgSocket = new ArrayList<Socket>();
private ArrayList<PrintWriter> printWriter =
            new ArrayList<PrintWriter>();
```

```
private ArrayList<BufferedReader> bufferedReader =
            new ArrayList<BufferedReader>();
private LinkedList<String> msgList = new LinkedList<String>();
private ArrayList<Socket> userSocket = new ArrayList<Socket>();
private ArrayList<PrintWriter> printUser = new ArrayList<PrintWriter>();
private ArrayList<ObjectInputStream> readUser =
            new ArrayList<ObjectInputStream>();
private ArrayList<User> userList = new ArrayList<User>();
// 定义一个全局布尔变量，控制线程是否运行，初始值为 true
private static boolean isRun = true;
```

2. 创建接收客户端发送的有关用户套接字的线程，代码如下所示：

```
// 接收客户端发送的有关用户套接字的线程
class AcceptUserSocketThread extends Thread {
    public void run() {
        while (isRun) {
            try {
                Socket socket = userServerSocket.accept();
                ObjectInputStream readerFromClient =
                    new ObjectInputStream(socket.getInputStream());
                PrintWriter printToClient =
                    new PrintWriter(socket.getOutputStream());
                printUser.add(printToClient);
                User user = (User) readerFromClient.readObject();
                if (user != null) {
                userList.add(user);
                // 在根节点下添加一个新节点
                root.add(new DefaultMutableTreeNode(
                    user.userName + ":"+ user.nickName));
                // 刷新树
                tree.setModel(model);
                tree.updateUI();
            // 调用 sendUserListToClient()方法向所有客户端发送目前所有用户信息
                sendUserListToClient();
                }
            } catch (Exception e) {
            e.printStackTrace();
            }
        }
    }
}
// 向所有客户端发送目前所有用户信息
```

```
private void sendUserListToClient() {
    String people = "";
    for (int i = 0; i < userList.size(); i++) {
        people += "&" + userList.get(i).nickName + "&"
            + userList.get(i).headImg;
    }
    for (int i = 0; i < printUser.size(); i++) {
        if (printUser.get(i) != null) {
            try {
                printUser.get(i).println(people);
                printUser.get(i).flush();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

3. 创建接收客户端发送的有关消息套接字的线程，代码如下所示：

```
// 接收客户端发送的有关消息套接字的线程
class AcceptMsgSocketThread extends Thread {
    public void run() {
        while (isRun) {
            try {
                Socket socket = msgServerSocket.accept();
                BufferedReader bufferedFromClient = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
                PrintWriter printWriterToClient = new PrintWriter(
                    socket.getOutputStream());
                printWriter.add(printWriterToClient);
                bufferedReader.add(bufferedFromClient);
                // 启动接收此客户发送聊天信息的线程
                new GetMessageThread(bufferedFromClient).start();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

4. 创建接收客户发送聊天信息的线程，代码如下所示：

```
// 接收客户发送聊天信息的线程
```

```
class GetMessageThread extends Thread {
    private String stringFromClient = null;
    private BufferedReader bufferedReaderFromClient;
    public GetMessageThread(BufferedReader buffer) throws Exception {
        bufferedReaderFromClient = buffer;
    }
    public void run() {
        try {
            do {
            stringFromClient = bufferedReaderFromClient.readLine();
                msgList.addFirst(stringFromClient);
            } while (!stringFromClient.toLowerCase().endsWith("bye"));
            int index = bufferedReader.indexOf(bufferedReaderFromClient);
            bufferedReader.remove(index);
            printWriter.remove(index);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5. 创建向所有客户端发送消息的线程，代码如下所示：

```
// 向所有客户端发送消息的线程
class SendMessageThread extends Thread {
    public SendMessageThread() throws Exception {
        super();
    }
    public void run() {
        while (isRun) {
            try {
                String s = null;
                if (!msgList.isEmpty()) {
                    s = (String) msgList.removeLast();
                    txtContent.append(s + "\n");
                    for (int i = 0; i < printWriter.size(); i++) {
                        if (printWriter.get(i) != null) {
                            printWriter.get(i).println(s);
                            printWriter.get(i).flush();
                        }
                    }
                } else {
                    sleep(100);
```

```
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

6. 在 ChatServer 类的构造方法中添加以下代码：

```
root = new DefaultMutableTreeNode("目前在线用户(用户名:昵称)");
    model = new DefaultTreeModel(root);
    tree = new JTree(model);
    spRight = new JScrollPane(tree);
    msgServerSocket = new ServerSocket(
        Integer.parseInt(Config.getValue("msgport")));
    userServerSocket = new ServerSocket(
        Integer.parseInt(Config.getValue("userport")));
    // 启动接收用户套接字的线程
    new AcceptUserSocketThread().start();
    // 启动接收消息套接字的线程
    new AcceptMsgSocketThread().start();
    // 启动发送消息的线程
    new SendMessageThread().start();
```

7. 实现单击“发送”按钮的事件处理。

ChatServer 类实现 ActionListener 接口，代码如下所示：

```
public class ChatServer extends JFrame implements ActionListener
```

在“发送”按钮上添加监听，代码如下所示：

```
btnSend.addActionListener(this);
```

编写事件处理方法，代码如下所示：

```
// 事件处理方法
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == btnSend) {
        String str = txtSend.getText();
        if (!str.equals("")) {
            msgList.addFirst("系统消息：" + str);
        }
    }
}
```

实践 5.G.4

升级聊天室客户端和服务器端的程序代码，具体要求如下：

- 当关闭聊天主窗口 FrmMain 时，同时关闭网络通信连接。
- 当关闭服务器 ChatServer 时，停止所有线程，并释放资源。

分析

1. 创建一个名为 WindowEventHandle 继承自 WindowAdapter 的窗口监听类，用于监听窗口事件。重写 windowClosing()事件处理方法。
2. 在窗口中添加 WindowEventHandle 监听对象。
3. 在 FrmMain.java 中增加 closeConnection()方法，用于关闭连接。
4. 在 ChatServer.java 中，将 isRun 设置为 false 停止线程；关闭所有服务套接字、流，并释放所有资源。

参考解决方案

1. 修改 FrmMain.java 程序，修改后的代码如下所示：

```
public class FrmMain extends JFrame implements ActionListener {
    private JSplitPane splitPaneV, splitPaneH;
    private JScrollPane spCenter;
    private JPanel pRight;
    private JPanel pdown;
    private JTextArea txtContent;
    private JLabel lblSend;
    private JTextField txtSend;
    private JButton btnSend;
    // 当前聊天用户的用户名
    public static String username = "";
    // 当前聊天用户的昵称
    public static String nickname = "";
    // 当前聊天用户的头像
    public static String headimg = "";
    // 用于消息传递的套接字
    private Socket socketMsg;
    // 用于用户传递的套接字
    private Socket socketUser;
    private PrintWriter printMsg;
    private BufferedReader readMsg;
    private ObjectOutputStream printUser;
    private BufferedReader readUser;
```

```
    private String strMsg = null;
    public FrmMain() {
        super("聊天室");
        txtContent = new JTextArea();
        txtContent.setEditable(false);
        spCenter = new JScrollPane(txtContent);
        pdown = new JPanel();
        lblSend = new JLabel("输入：");
        txtSend = new JTextField(20);
        btnSend = new JButton("发送");
        pdown.add(lblSend);
        pdown.add(txtSend);
        pdown.add(btnSend);
        btnSend.addActionListener(this);
        pRight = new JPanel(null);
        splitPaneV = new JSplitPane(JSplitPane.VERTICAL_SPLIT, spCenter, pdown);
        splitPaneV.setDividerLocation(320);
        splitPaneV.setResizeWeight(1);
        splitPaneH = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, splitPaneV,
                pRight);
        splitPaneH.setDividerLocation(350);
        splitPaneH.setResizeWeight(1);
        this.add(splitPaneH);
        this.setSize(500, 400);
        this.setLocation(300, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addWindowListener(new WindowEventHandle());
        createUserListConnection();
        createMsgConnection();
        new GetUserListFromServer().start();
        new GetMsgFromServer().start();
    }
    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == btnSend) {
            String str = txtSend.getText();
            if (!str.equals("")) {
                // 网络通信
                printMsg.println(nickname + ":" + str);
                printMsg.flush();
                txtSend.setText("");
            }
        }
```

```
    }
    // 窗体监听类
    class WindowEventHandle extends WindowAdapter {
        // 当窗体关闭时，同时关闭网络通信连接
        public void windowClosing(WindowEvent e) {
            closeConnection();
        }
    }
    // 关闭网络通信连接
    private void closeConnection() {
        try {
            printMsg.println("bye");
            printMsg.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 建立聊天信息链接
    private void createMsgConnection() {
        try {
            socketMsg = new Socket(Config.getValue("server"),
                Integer.parseInt(Config.getValue("msgport")));
            printMsg = new PrintWriter(socketMsg.getOutputStream(), true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 建立用户信息链接
    private void createUserListConnection() {
        try {
            socketUser = new Socket(Config.getValue("server"),
                Integer.parseInt(Config.getValue("userport")));
            printUser = new ObjectOutputStream(socketUser.getOutputStream());
            // 封装当前用户的信息
            User user = new User(username, nickname, headimg);
            // 将当前用户发送给服务器
            printUser.writeObject((User) user);
            printUser.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 从服务器获取聊天信息的线程
```

```
class GetMsgFromServer extends Thread {
    public GetMsgFromServer() {
        super();
    }
    public void run() {
        try {
            readMsg = new BufferedReader(new InputStreamReader(
                socketMsg.getInputStream()));
            do {
                strMsg = readMsg.readLine();
                // 在文本域中显示聊天信息
                txtContent.append(strMsg + "\n");
            } while (strMsg != null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
// 从服务器获取所有聊天用户信息的线程
class GetUserListFromServer extends Thread {
    public GetUserListFromServer() {
        super();
    }
    public void run() {
        try {
            readUser = new BufferedReader(new InputStreamReader(
                socketUser.getInputStream()));
            do {
                String line = readUser.readLine();
                if (line != null) {
                    StringTokenizer st = new StringTokenizer(line, "&");
                    pRight.removeAll();
                    int y = 10;
                    while (st.hasMoreTokens()) {
                        String name = st.nextToken();
                        String img = st.nextToken();
                        System.out.println(name + ":" + img);
                        JLabel userNode = new JLabel(name, new ImageIcon(
                                "images/" + img), JLabel.LEFT);
                        userNode.setBounds(10, y, 140, 40);
                        pRight.add(userNode);
                        y += 45;
                    }
```

```
                        pRight.setSize(150, 300);
                        splitPaneH.setDividerLocation(350);
                        FrmMain.this.setSize(0, 0);
                        FrmMain.this.setSize(500, 400);
                    }
                } while (true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. 修改 ChatServer.java 程序，修改后的代码如下所示：

```
public class ChatServer extends JFrame implements ActionListener {
    private JSplitPane splitPaneV, splitPaneH;
    private JScrollPane spCenter, spRight;
    private JPanel pdown;
    private JTextArea txtContent;
    private JLabel lblSend;
    private JTextField txtSend;
    private JButton btnSend;
    private DefaultMutableTreeNode root;
    private DefaultTreeModel model;
    private JTree tree;
    // 消息服务套接字
    private ServerSocket msgServerSocket;
    // 用户服务套接字
    private ServerSocket userServerSocket;
    private ArrayList<Socket> msgSocket = new ArrayList<Socket>();
    private ArrayList<PrintWriter> printWriter = new ArrayList<PrintWriter>();
    private ArrayList<BufferedReader> bufferedReader =
                new ArrayList<BufferedReader>();
    private LinkedList<String> msgList = new LinkedList<String>();
    private ArrayList<Socket> userSocket = new ArrayList<Socket>();
    private ArrayList<PrintWriter> printUser = new ArrayList<PrintWriter>();
    private ArrayList<ObjectInputStream> readUser =
                new ArrayList<ObjectInputStream>();
    private ArrayList<User> userList = new ArrayList<User>();
    // 定义一个全局布尔变量，控制线程是否运行，初始值为 true
    private static boolean isRun = true;
    public ChatServer() throws Exception {
        super("服务器");
```

```
        txtContent = new JTextArea();
        txtContent.setEditable(false);
        spCenter = new JScrollPane(txtContent);
        pdown = new JPanel();
        lblSend = new JLabel("系统消息：");
        txtSend = new JTextField(20);
        btnSend = new JButton("发送");
        pdown.add(lblSend);
        pdown.add(txtSend);
        pdown.add(btnSend);
        btnSend.addActionListener(this);
        root = new DefaultMutableTreeNode("目前在线用户(用户名:昵称)");
        model = new DefaultTreeModel(root);
        tree = new JTree(model);
        spRight = new JScrollPane(tree);
        splitPaneV = new JSplitPane(JSplitPane.VERTICAL_SPLIT, spCenter, pdown);
        splitPaneV.setDividerLocation(420);
        splitPaneV.setResizeWeight(1);
        splitPaneH = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, splitPaneV,
                spRight);
        splitPaneH.setDividerLocation(400);
        splitPaneH.setResizeWeight(1);
        this.add(splitPaneH);
        this.setSize(600, 500);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addWindowListener(new WindowEventHandle());
        msgServerSocket = new ServerSocket(
            Integer.parseInt(Config.getValue("msgport")));
        userServerSocket = new ServerSocket(
            Integer.parseInt(Config.getValue("userport")));
        // 启动接收用户套接字的线程
        new AcceptUserSocketThread().start();
        // 启动接收消息套接字的线程
        new AcceptMsgSocketThread().start();
        // 启动发送消息的线程
        new SendMessageThread().start();
    }
    // 事件处理方法
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnSend) {
            String str = txtSend.getText();
            if (!str.equals("")) {
                msgList.addFirst("系统消息：" + str);
            }
```

```
        }
    }
    // 接收客户端发送的有关用户套接字的线程
    class AcceptUserSocketThread extends Thread {
        public void run() {
            while (isRun) {
                try {
                    Socket socket = userServerSocket.accept();
                    ObjectInputStream readerFromClient = new ObjectInputStream(
                            socket.getInputStream());
                    PrintWriter printToClient = new PrintWriter(
                        socket.getOutputStream());
                    printUser.add(printToClient);
                    User user = (User) readerFromClient.readObject();
                    if (user != null) {
                        userList.add(user);
                        // 在根节点下添加一新节点
                        root.add(new DefaultMutableTreeNode(user.userName + ":"
                            + user.nickName));
                        // 刷新树
                        tree.setModel(model);
                        tree.updateUI();
                // 调用 sendUserListToClient()方法向所有客户端发送目前所有用户信息
                        sendUserListToClient();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
    // 向所有客户端发送目前所有用户信息
    private void sendUserListToClient() {
        String people = "";
        for (int i = 0; i < userList.size(); i++) {
            people += "&" + userList.get(i).nickName + "&"
                    + userList.get(i).headImg;
        }
        for (int i = 0; i < printUser.size(); i++) {
            if (printUser.get(i) != null) {
                try {
                    printUser.get(i).println(people);
                    printUser.get(i).flush();
                } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    }
}
// 接收客户端发送的有关消息套接字的线程
class AcceptMsgSocketThread extends Thread {
    public void run() {
        while (isRun) {
            try {
                Socket socket = msgServerSocket.accept();
                BufferedReader bufferedFromClient = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
                PrintWriter printWriterToClient = new PrintWriter(
                    socket.getOutputStream());
                printWriter.add(printWriterToClient);
                bufferedReader.add(bufferedFromClient);
                // 启动接收此客户发送聊天信息的线程
                new GetMessageThread(bufferedFromClient).start();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
// 接收客户发送聊天信息的线程
class GetMessageThread extends Thread {
    private String stringFromClient = null;
    private BufferedReader bufferedReaderFromClient;
    public GetMessageThread(BufferedReader buffer) throws Exception {
        bufferedReaderFromClient = buffer;
    }
    public void run() {
        try {
            do {
                stringFromClient = bufferedReaderFromClient.readLine();
                msgList.addFirst(stringFromClient);
            } while (!stringFromClient.toLowerCase().endsWith("bye"));
            int index = bufferedReader.indexOf(bufferedReaderFromClient);
            bufferedReader.remove(index);
            printWriter.remove(index);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        }
    }
    // 向所有客户端发送消息的线程
    class SendMessageThread extends Thread {
        public SendMessageThread() throws Exception {
            super();
        }
        public void run() {
            while (isRun) {
                try {
                    String s = null;
                    if (!msgList.isEmpty()) {
                        s = (String) msgList.removeLast();
                        txtContent.append(s + "\n");
                        for (int i = 0; i < printWriter.size(); i++) {
                            if (printWriter.get(i) != null) {
                                printWriter.get(i).println(s);
                                printWriter.get(i).flush();
                            }
                        }
                    } else {
                        sleep(100);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
    // 窗体监听类
    class WindowEventHandle extends WindowAdapter {
        // 当窗体关闭时，停止所有通信线程
        public void windowClosing(WindowEvent e) {
            isRun = false;
            try {
                msgServerSocket.close();
                userServerSocket.close();
                for (int i = 0; i < msgSocket.size(); i++) {
                    msgSocket.get(i).close();
                }
                for (int i = 0; i < printWriter.size(); i++) {
                    printWriter.get(i).close();
                }
                for (int i = 0; i < bufferedReader.size(); i++) {
                    bufferedReader.get(i).close();
```

```
                }
                for (int i = 0; i < userSocket.size(); i++) {
                    userSocket.get(i).close();
                }
                for (int i = 0; i < printUser.size(); i++) {
                    printUser.get(i).close();
                }
                for (int i = 0; i < readUser.size(); i++) {
                    readUser.get(i).close();
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            } finally {
                msgServerSocket = null;
                userServerSocket = null;
                msgSocket = null;
                printWriter = null;
                bufferedReader = null;
                msgList = null;
                userSocket = null;
                printUser = null;
                readUser = null;
                userList = null;
            }
        }
    }
    public static void main(String args[]) throws Exception {
        ChatServer chatServer = new ChatServer();
        chatServer.setVisible(true);
    }
}
```

3. 启动服务器应用程序和客户端应用程序，进行聊天，如图 5.2 和图 5.3 所示：

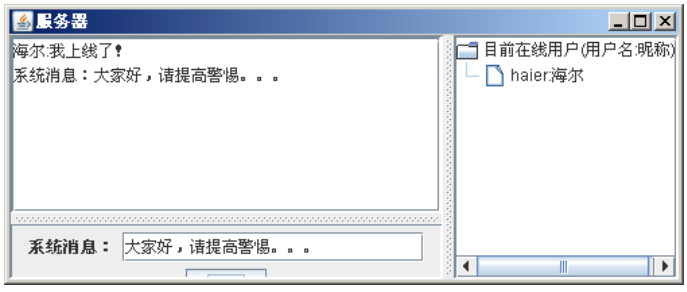


图 5.2　服务器界面



图 5.3 客户端界面

知识拓展

1. 2D 绘图

Java 2D API 由 Sun 公司与 Adobe 系统公司合作推出，提供了一个功能强大且灵活的二维图形框架。Java 2D API 扩展了 java.awt 包中定义的 Graphics 类和 Image 类，提供了高性能的二维图形、图像和文字，同时又维持了对现有 AWT 应用的兼容。

2D 绘图时需要产生一个 Graphics2D 对象，设定所要的状态属性，再调用 Graphics2D 所提供的方法完成整个绘图工作。

Graphics2D 类扩展 Graphics 类，提供对几何形状、坐标转换、颜色管理和文本布局更为复杂的控制。它是用于在 Java(tm)平台上呈现二维形状、文本和图像的基础类。

Graphics2D 常用的两个画图的方法如下：

- draw(Shape s)：勾画图形对象的轮廓。
- fill(Shape s)：绘画图形并填充图形的内部区域。

下面是一个 2D 绘图的简单示例，代码如下所示：

```
public class DrawDemo extends JFrame {
    public DrawDemo() {
        super("2D 绘图");
        // 创建绘图面板
        DrawPane map = new DrawPane();
        this.add(map);
        this.setSize(400,300);
        this.setLocation(300,300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] arg) {
        DrawDemo frame = new DrawDemo();
        frame.setVisible(true);
    }
```

```
}
// 带绘图的面板
class DrawPane extends JPanel {
    // 重写 paintComponent()方法，该方法将传入一个 Graphics 对象（画笔），可以在 JPanel 容
    // 器中画图
    public void paintComponent(Graphics g) {
        // 将 Graphics 对象转换成 Graphics2D
        Graphics2D g2D = (Graphics2D) g;
        // 设置画笔的颜色为红色
        g2D.setColor(Color.red);
        // 在（30,80）位置处开始写一个内容为"画空心矩形"的字符串
        g2D.drawString("画空心矩形", 30, 80);
        // 创建一个矩形图形对象，矩形左上角坐标（30，100），宽度 80，高度 40
        Rectangle2D.Double rect = new Rectangle2D.Double(30, 100, 80, 40);
        // 勾画矩形轮廓（空心矩形）
        g2D.draw(rect);
        g2D.setColor(Color.blue);
        g2D.drawString("画实心椭圆", 140, 80);
        // 创建一个椭圆图形对象
        Ellipse2D.Double elip = new Ellipse2D.Double(140, 100, 80, 40);
        // 画实心椭圆
        g2D.fill(elip);
    }
}
```

在上述代码中，在容器中绘图需要重写组件的 paintComponent()方法，该方法将传入一个 Graphics 对象，此对象不必人为创建，当创建容器组件对象时自动实例化 Graphics 对象并传给 paintComponent()方法。

运行结果如图 5.4 所示。



图 5.4　2D 绘图



拓展练习

练习 5.E.1

使用 2D 绘图画如图 5.5 所示的图形。

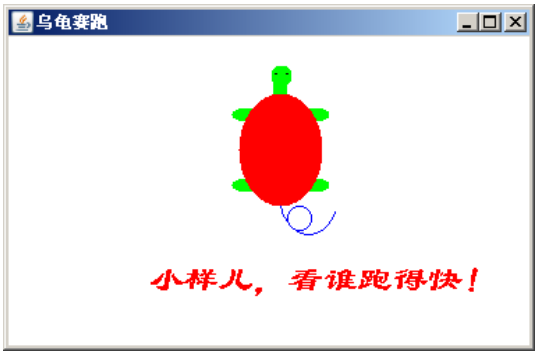


图 5.5　乌龟赛跑

实践 6　JDBC

实践指导

实践 6.G.1

在 Windows XP 下，手动建立 ODBC，并通过 JDBC 进行连接测试。

分析

1. 使用 ODBC 时，ODBC 驱动程序管理器与驱动程序都必须手工装入到每台客户机系统中，Windows XP 自带 ODBC 驱动。
2. Java 中提供了 JDBC/ODBC 桥接驱动以通过 JDBC API 使用 ODBC 的功能。
3. 在此将配置 ODBC 到 Oracle10g 数据库的连接，并使用理论篇中搭建的 Student 数据表进行测试。

参考解决方案

1. 建立 ODBC 数据源

选择“开始”→“设置”→“控制面板”命令，在弹出的控制面板界面中，选择“管理工具”→“数据源（ODBC）”，可以看到“ODBC 数据源管理器”对话框，如图 6.1 所示。



图 6.1　ODBC 数据源管理器

在“用户 DSN”选项卡界面中显示了已有的数据源名称。单击“添加”按钮，弹出“创建新数据源”对话框，如图 6.2 所示。

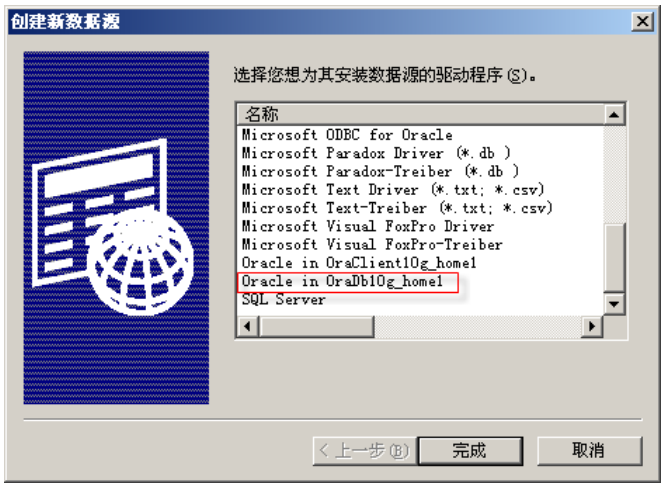


图 6.2　“创建新数据源”对话框

拖动滚动条，选择“Oracle in OraDb10g_home1”项（出现此项的前提是本机安装了 Oracle 服务器程序，如果只安装了 Oracle 客户端，可以选择“Oracle in OraClient10g_home1”，并配置客户端到 Oracle 服务器的连接即可），单击“完成”按钮，进入“Oracle ODBC Driver Configuration”对话框，如图 6.3 所示。

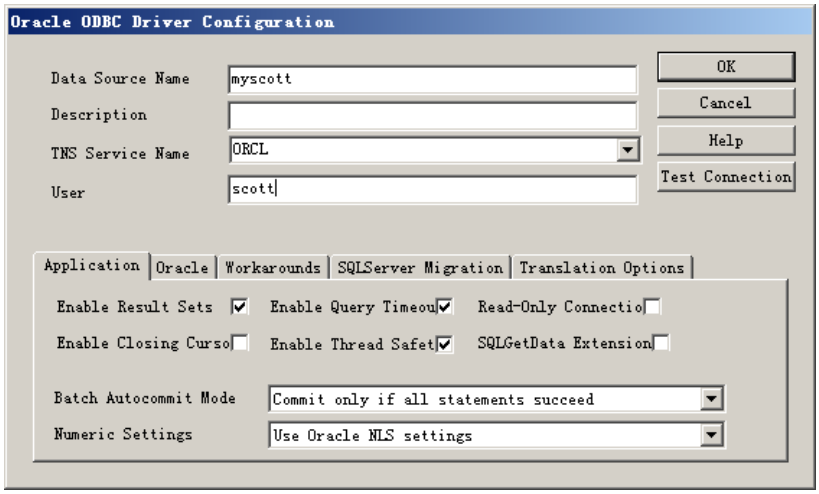


图 6.3　“Oracle ODBC Driver Configuration”对话框

在 Data Source Name 中输入数据源的名称“myscott”，在 TNS Service Name 下拉列表中选择“ORCL”（需要连接的 Oracle 数据库实例），User 文本框中输入 scott，其他选项保持默认即可。单击“Test Connection”按钮，弹出“Oracle ODBC Driver Connect”对话框，如图 6.4 所示。

在此对话框中输入 scott 用户的密码：tiger，单击“OK”按钮，如果提供的连接信息无误，则弹出如图 6.5 所示的对话框，提示测试通过。

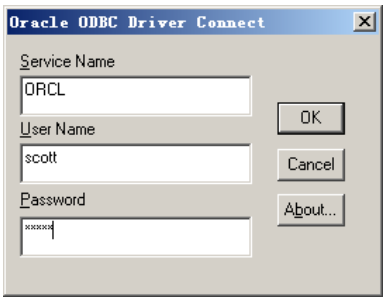


图 6.4 “Oracle ODBC Driver Connect”对话框



图 6.5 通过测试

再次回到“ODBC 数据源管理器”对话框，在“用户 DSN”选项卡界面中出现了刚才创建的“myscott”数据源。

2. 编写测试代码

为方便实现，使用理论篇设计的常量类和工具类，由于为了通过 ODBC 连接，需调整常量类的连接信息，代码如下：

```
public class Constants {
    // 使用 jdbc/odbc 桥接驱动
    public static String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
    // 使用 odbc 协议
    public static String URL = "jdbc:odbc:myscott";
    public static String USERNAME = "scott";
    public static String PASSWORD = "tiger";
}
```

工具类 DBUtil.java 无需任何调整。

测试代码可以直接使用理论篇的 ConnectionDemo.java，如果运行通过将出现如下结果：

```
连接成功!
查询成功!
95001
95002
95003
95005
```

实践 6.G.2

基于实践 5 中的聊天室程序，使用 JDBC 访问 Oracle 数据库，具体要求如下：

- 将用户注册信息存入 Oracle 数据库的 userdetails 表中。
- 登录时，查询 userdetails 表，对用户信息进行验证。验证通过才能进入聊天室主窗口，否则提示错误信息。

分析

1. 在聊天室程序中，用户注册信息用于用户信息的保存和登录时用户信息的验证。
2. 针对当前应用，对于数据库的操作主要集中在数据的添加（存储用户信息）和查询（验证用户信息）。

参考解决方案

1. 首先使用 Scott 用户创建用户信息表，代码如下：

```
CREATE TABLE userdetails(
username VARCHAR2(20) primary key,--用户名
nickname VARCHAR2(20) NOT NULL,--昵称
password VARCHAR2(20) NOT NULL,--密码
sex char(1) NOT NULL,--0:男，1：女
headimg VARCHAR2(20) NOT NULL--头像
)
```

2. 为方便管理，基于实践 5，将数据库的连接信息放入 Config.ini 文件中，代码如下：

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:orcl
username=scott
password=tiger
```

3. 调整 DBUtil（数据库工具类）中的数据库连接代码，并对查询操作和常用 DML 操作进行抽象定义，代码如下：

```
public class DBUtil {
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    public Connection getConnection()
    throws ClassNotFoundException, SQLException,
            InstantiationException, IllegalAccessException {
        String DRIVER = Config.getValue("driver");
        String URL = Config.getValue("url");
        String USERNAME = Config.getValue("username");
        String PASSWORD =Config.getValue("password");
        System.out.println(DRIVER);
        try {
            // 指定驱动程序
            Class.forName(DRIVER);
            // 建立数据库连接
```

```
            conn = DriverManager.getConnection(URL,USERNAME, PASSWORD);
            return conn;
        } catch (Exception e) {
            // 如果连接过程出现异常，抛出异常信息
            throw new SQLException("驱动错误或连接失败！");
        }
    }
    // 可以根据传入的查询语句和参数，自动构造 PreparedStatement，进行查询操作
    public ResultSet executeQuery(String preparedSql, String[] param) {
        // 处理 SQL,执行 SQL
        try {
            // 得到 PreparedStatement 对象
            pstmt = conn.prepareStatement(preparedSql);
            if (param != null) {
                for (int i = 0; i < param.length; i++) {
                    // 为预编译 SQL 设置参数
                    pstmt.setString(i + 1, param[i]);
                }
            }
            // 执行 SQL 语句
            rs = pstmt.executeQuery();
        } catch (SQLException e) {
            // 处理 SQLException 异常
            e.printStackTrace();
        }
        return rs;
    }
    // 可以根据传入的查询语句和参数，自动构造 PreparedStatement，进行 DML 操作
    public int executeUpdate(String preparedSql, String[] param) {
        int num = 0;
        // 处理 SQL，执行 SQL
        try {
            // 得到 PreparedStatement 对象
            pstmt = conn.prepareStatement(preparedSql);
            if (param != null) {
                for (int i = 0; i < param.length; i++) {
                    // 为预编译 SQL 设置参数
                    pstmt.setString(i + 1, param[i]);
                }
            }
            // 执行 SQL 语句
            num = pstmt.executeUpdate();
        } catch (SQLException e) {
```

```
            // 处理 SQLException 异常
            e.printStackTrace();
        }
        return num;
    }
    //关闭数据库，释放资源
    public void closeAll()  {
        // 如果 rs 不为空，关闭 rs
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        // 如果 pstmt 不为空，关闭 pstmt
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        // 如果 conn 不为空，关闭 conn
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4. 应用新调整的 DBUtil 工具类，为注册窗体（FrmRegist）的“确定”按钮添加事件处理，实现用户注册操作。在 4.G.2 中 FrmRegist.java 代码的基础上，先增加一个注册用户信息的方法 registUser()，该方法的代码如下：

```
private boolean registUser(String username, String nickname,
        String password, String sex, String headimg) {
    boolean r = false;
    DBUtil db = new DBUtil();
    try {
```

```
        db.getConnection();
        String sql = "INSERT INTO userdetails
         (username,nickname,password,sex,headimg) values(?,?,?,?,?)";
        int rs = db.executeUpdate(sql, new String[] { username, nickname,
                password, "" + sex, headimg });
        if (rs > 0) {// 插入成功
            r = true;
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // 关闭数据库连接
        db.closeAll();
    }
    return r;
}
```

在 FrmRegist.java 代码中查找 btnOkClick()方法，将该方法中的下面语句：

```
if (userpwd.equals("haiersoft"))
```

改为

```
if (registUser(username, nickname, userpwd, sex, checkedHeadImg))
```

调用 registUser()方法并传入参数，将用户注册信息插入到数据库中。

5. 为登录窗体（FrmLogin）的“确定”按钮添加事件处理，实现用户登录操作，验证用户信息。在 4.D.1 中 FrmLogin.java 代码的基础上，先增加一个验证用户信息的方法 check()，该方法的代码如下：

```
private boolean check(String uname, String upwd) {
    boolean r = false;
    DBUtil db = new DBUtil();
    try {
        // 连接数据库
        db.getConnection();
        String sql = "SELECT nickname,headimg FROM userdetails
                    WHERE username=? AND password=?";
        ResultSet rs = db.executeQuery(sql, new String[] { uname, upwd });
        // 合法的用户
        if (rs != null && rs.next()) {
            r = true;
```

```
            // 将用户的昵称、头像信息存储到聊天主窗口中
            FrmMain.username = uname;
            FrmMain.nickname = rs.getString("nickname");
            FrmMain.headimg = rs.getString("headimg");
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // 关闭数据库连接
        db.closeAll();
    }
    return r;
}
```

在 FrmLogin.java 代码中查找 btnOkClick()方法，将该方法中的下面语句：

```
if (userpwd.equals("haiersoft"))
```

改为

```
if (check(username, userpwd))
```

调用 check()方法并传入参数，对用户登录信息进行验证。

6. 测试：测试时务必要在 config/config.ini 文件中指明数据库连接的具体信息。



知识拓展

1. 可滚动和可更新的结果集

默认情况下，ResultSet 的游标只能使用 next()方法往前移动。在 JDBC 2.0 后的版本中，对 ResultSet 作了重大的改进：让 ResultSet 的游标可以随意移动，并且还可以对数据库进行增加、删除及修改操作。

要让 ResultSet 可以滚动和更新，必须在创建 Statement 对象的时候指定对应的参数：

```
Statement stmt = conn.createStatement(type,concurrency);
```

对于 PreparedStatement，使用下面的方式指定参数：

```
PreparedStatement pstmt = conn.prepareStatement(sql,type,concurrency);
```

其中，type 表示 ResultSet 的类型，而 concurrency 表示是否可以使用 ResultSet 来更新数

据库。

type 及 concurrency 的取值及作用见表 6.1。

表 6.1　type 及 concurrency 的取值及作用

类型	取值	功能说明
type	ResultSet.TYPE_FORWARD_ONLY	结果集不能滚动，这是默认值
	ResultSet.TYPE_SCROLL_INSENSITIVE	结果集可以滚动，但 ResultSet 对数据库中发生的数据改变不敏感
	ResultSet.TYPE_SCROLL_SENSITIVE	结果集可以滚动，并且 ResultSet 对数据库中发生的数据改变敏感
concurrency	ResultSet.CONCUR_READ_ONLY	只读结果集，不能用于更新数据库
	ResultSet.CONCUR_UPDATABLE	可更新结果集，可以用于更新数据库

当使用 TYPE_SCROLL_INSENSITIVE 或者 TYPE_SCROLL_SENSITIVE 来创建 Statement 对象时，可以使用 ResultSet 的 first、last、beforeFirst、afterLast、relative、absolute 等方法在结果集中随意移动游标。

下述代码演示了如何创建并使用可滚动的结果集。

```
public class RollbackResultDemo {
    // 输出当前行信息
    public static void print(ResultSet rs) throws SQLException {
        System.out.println("行 " + rs.getRow() + ":" + rs.getString(1) + ","
                + rs.getString(2) + "," + rs.getString(3));
    }
    public static void main(String args[]) throws ClassNotFoundException {
        try {
            Connection conn = DBUtil.getConnection();
            // 创建可滚动的 Statment 对象
            Statement stmt = conn.createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            // 获取查询结果集
            ResultSet rs = stmt.executeQuery("SELECT sno,name,age FROM student");
            while (rs.next()) {
                print(rs);
            }
            // 第一行
            rs.first();
            print(rs);
            // 最后一行
            rs.last();
            print(rs);
```

```
            // 第二行
            rs.absolute(2);
            print(rs);
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
行 1:95001,张飞,23
行 2:95002,关羽,23
行 3:95003,貂蝉,23
行 4:95005,小乔,23
行 1:95001,张飞,23
行 4:95005,小乔,23
行 2:95002,关羽,23
```

要让结果集成为可以更新数据库的结果集，必须在创建 Statement 的时候指定第二个参数为 ResultSet.CONCUR_UPDATABLE。如果结果集是可更新的，可调用 ResultSet 相关方法进行更新操作，其常用更新方法及使用说明见表 6.2。

表 6.2　ResultSet 的更新方法列表

方法名	功能说明
void moveToCurrentRow()	将游标移动到记住的游标位置，通常为当前行
void moveToInsertRow()	将游标移动到插入行
void cancelRowUpdates()	取消当前行的更新信息
void insertRow()	将插入行的内容插入到结果集并同步到数据库中
void deleteRow()	从结果集和底层数据库中删除当前行
void updateRow()	用结果集的当前行的新内容更新底层数据库
void updateString(int x, String y)	用 String 值更新 x 列值
void updateInt(int x, int y)	用 int 值更新 x 列值

有几个方法需要注意：

updateXXX()方法和 setXXX()方法类似，XXX 为数据类型。updateXXX()方法接收字段索引和字段名称，可通过字段的索引进行更新，也可以指定字段名称进行更新。需要注意的是，使用 updateXXX()方法只会修改结果集中的数据，要让修改真正影响到数据库，还必须在执行 updateXXX()方法后执行其他的方法。

updateRow()方法用于将当前行中的数据更新同步到数据库中，这个方法要紧跟着

updateXXX()执行，否则如果执行完 updateXXX()方法后移动了游标，将会取消对结果集所作的更新。

moveToInsertRow()方法将指针移动到特定的位置，在这个特定的位置上可以进行操作，这个位置称做“插入行”。在将指针移动到插入行后，需要调用 updateXXX()方法来向这个当前为空的行中插入数据。注意，在执行完 updateXXX()方法的时候，需要调用 insertRow()方法来将新增到结果集中的数据同步到数据库中。

下述代码演示了如何创建并使用可编辑的结果集。

```
public class EditableResultSetDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        try {
            Connection conn = DBUtil.getConnection();
            // 创建可滚动的 Statment 对象
            Statement stmt = conn.createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
            // 获取查询结果集
            ResultSet rs = stmt.executeQuery("SELECT sno,name,age FROM student");
            int count = 0;
            // 对偶数行的年龄字段更新为 25
            while (rs.next()) {
                count++;
                if (count % 2 == 0) {
                    rs.updateInt(3, 25);
                    rs.updateRow();
                }
            }
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

上述代码对查询结果集中的偶数行数据的年龄字段进行了调整，统一设置为 25。执行完毕可通过查询操作查看修改结果。

2. 操作 BLOB/CLOB 数据

在 Oracle9i 以后的版本中，新增了两种数据类型：用于保存二进制数据的 BLOB（Binary Large OBject）及用于保存大量文本数据的 CLOB（Character Large OBject）。使用这两种数据类型可以将图片等内容保存到数据库中，或者将一个内容较大的文本文件保存到数据库中。下面通过对 BLOB 类型数据的存取，讲解此两种数据类型的使用方法。

如果需要将图片保存到数据库中，可以使用 PreparedStatement 的如下方法：

```
setBinaryStream(int idx,InputStream is,int length)
```

或者：

```
setBlob(int idx,Blob b)
```

从数据库中读出 BLOB 的数据，可以使用 ResultSet 中的如下方法：

```
InputStream getBinaryStream(int columnIndex)
```

或者：

```
Blob getBlob(int columnIndex)
```

下面的代码通过图片文件在 Oracle 中的存取，演示对 BLOB 数据的操作。

首先需创建能够存储图片的表格，执行如下脚本：

```
CREATE TABLE picture(
pid VARCHAR2(5) PRIMARY KEY,--图片 id
filename VARCHAR2(20) NOT NULL,--图片名称
pic  LONG RAW NOT NULL --图片内容);
```

注意　如果要把一个二进制文件存入 Oracle，用标准的 JDBC 就要用 LONG RAW 类型，如果一定要用 BLOB 存储，就必须用 Oracle 自己的方法。

在 Oracle 中进行文件存取，代码如下：

```
public class BlobDemo {
    // 存储图片
    public static void savePic() {
        Connection conn = null;
        FileInputStream fis = null;
        try {
            conn = DBUtil.getConnection();
            String insertStr = "insert into picture values(?,?,?)";
            PreparedStatement stmt = conn.prepareStatement(insertStr);
            stmt.setString(1, "A0001");
            stmt.setString(2, "test.jpg");
            // 获取图片文件
            File file = new File("test.jpg");
            // 取得文件流
            fis = new FileInputStream(file);
            // 将二进制流数据传入 PreparedStatement 对象
            stmt.setBinaryStream(3, fis, (int) file.length());
            stmt.executeUpdate();
        } catch (SQLException sqle) {
            System.out.println(sqle);
```

```
        } catch (IOException ioe) {
            System.out.println(ioe);
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if (conn != null) {
                try {
                    // 关闭连接
                    DBUtil.closeConnection(conn);
                } catch (SQLException se) {
                    se.printStackTrace();
                }
            }
            if (fis != null) {
                try {
                    // 关闭文件流
                    fis.close();
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
            }
        }
    }
    // 读取图片
    public static void getPic() {
        Connection conn = null;
        FileOutputStream fos = null;
        try {
            conn = DBUtil.getConnection();
            String selectStr = "select pid,filename,pic from picture";
            PreparedStatement stmt = conn.prepareStatement(selectStr);
            ResultSet rs = stmt.executeQuery();
            File file = new File("C:\\test_new.jpg");
            // 取得文件流
            fos = new FileOutputStream(file);
            if (rs.next()) {
                // 取得二进制数据流对象
                InputStream is = rs.getBinaryStream(3);
                int len = 0;
                byte b[] = new byte[4 * 1024];
                // 循环写文件
                while ((len = is.read(b)) != -1) {
                    fos.write(b, 0, len);
                }
                fos.flush();
                is.close();
```

```
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (conn != null) {
                try {
                    // 关闭连接
                    DBUtil.closeConnection(conn);
                } catch (SQLException se) {
                    se.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    // 关闭文件流
                    fos.close();
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
            }
        }
    }
    public static void main(String[] args) {
        savePic();
        System.out.println("向数据库写图片数据完毕！");
        getPic();
        System.out.println("从数据库读图片数据完毕！");
    }
}
```

如果运行没有异常，将在 C 盘根目录下生成 test_new.jpg 文件。

3. 批处理

所谓批处理，就是将很多 SQL 语句集合到一起执行，以提高效率。

要使用 JDBC 的批处理，需要使用 Statement 对象的 addBatch()方法向其中加入 SQL 语句，然后使用 Statement 对象的 executeBatch()方法来一次执行加入到批处理中的 SQL 语句。注意，在使用 addBatch()方法加入 SQL 语句的时候，语句并不会执行，只有在调用了 executeBatch()方法后，才会执行。批处理和事务处理通常会结合起来使用。

下面的代码通过两次向 Student 表中添加数据，演示 JDBC 批处理的使用。

```
public class BatchDemo {
    public static void main(String args[]) throws ClassNotFoundException {
        String selectSql = "SELECT sno,name,age FROM student";
        String batchSql1 = "INSERT INTO student VALUES('95008','周瑜',19,1)";
```

```
        String batchSql2 = "INSERT INTO student VALUES('95009','诸葛亮',21,1)";
        Connection conn = null;
        try {
            conn = DBUtil.getConnection();
            Statement stmt = conn.createStatement();
            // 获取并记录事务提交状态
            boolean autoCommit = conn.getAutoCommit();
            conn.setAutoCommit(false);
            // 添加批处理语句
            stmt.addBatch(batchSql1);
            stmt.addBatch(batchSql2);
            // 执行批处理
            stmt.executeBatch();
            conn.commit();
            // 恢复原有事务提交状态
            conn.setAutoCommit(autoCommit);
            // 执行查询，验证结果
            ResultSet rs = stmt.executeQuery(selectSql);
            while (rs.next()) {
                System.out.println("行 " + rs.getRow() + ":" + rs.getString(1)
                    + "," + rs.getString(2) + "," + rs.getString(3));
            }
            // 关闭连接
            DBUtil.closeConnection(conn);
        } catch (Exception e) {
            // 如果出现异常，回滚
            if(conn!=null){
                try{
                    conn.rollback();
                }
                catch(SQLException se){
                    se.printStackTrace();
                }
            }
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
行 1:95001,张飞,23
行 2:95002,关羽,25
行 3:95003,貂蝉,23
行 4:95005,小乔,25
行 5:95008,周瑜,19
行 6:95009,诸葛亮,21
```

从执行结果可以看出，两条插入语句都已顺利执行。

4. 使用 JDBC 连接不同的数据库

大多数的数据库厂商都提供了专用的 JDBC 驱动程序用于数据库的连接，在此只给出这些数据库使用专用 JDBC 驱动程序连接数据库的主要代码。详细的配置和使用方法请读者参阅相关资料进行学习。

1. Microsoft SQL Server 2000

包名：msbase.jar、mssqlserver.jar、msutil.jar

连接代码：

```
Class.forName( "com.microsoft.jdbc.sqlserver.SQLServerDriver" );
Connection conn =DriverManager.getConnection(
"jdbc:microsoft:sqlserver:// IP:1433;databaseName=db", username,password);
```

2. Microsoft SQL Server 2005

包名：sqljdbc.jar

连接代码：

```
Class.forName( " com.microsoft.sqlserver.jdbc.SQLServerDriver" );
Connection conn =DriverManager.getConnection(
" jdbc:sqlserver:// IP:1433;databaseName=db", username,password);
```

3. MySQL

包名：mm.mysql-3.0.2-bin.jar

连接代码：

```
Class.forName( "org.gjt.mm.mysql.Driver" );
Connection conn = DriverManager.getConnection( "jdbc:mysql:// IP:3306/数据库名",
username,password);
```

4. Sybase

包名：jconn2.jar

连接代码：

```
Class.forName( "com.sybase.jdbc2.jdbc.SybDriver" );
Connection conn = DriverManager.getConnection( "jdbc:sybase:Tds: IP:2638",
username,password);
```

5. DB2

连接代码：

```
Class.forName("com.ibm.db2.jdbc.net.DB2Driver");
Connection conn = DriverManager.getConnection("jdbc:db2: IP:6789 ", username,
```

```
password);
```



拓展练习

练习 6.E.1

编写程序，连接 SQL 2005 数据库，查询数据库中 pubs 表的所有数据，并以 JTable 的形式显示到界面中。

练习 6.E.2

在 MySQL 中创建一个名为 Haier 的数据库，在该数据库中创建一张 Userdetails 表，表有三个字段（用户名、密码、权限）。编写一个应用程序，插入数据到 Userdetails 表中。

实践 7　RMI



实践指导

实践 7.G.1

某银行在全球的主要城市都有部门，但只有一个中央数据库。此银行要创建一个让它的客户可以打开银行账户的应用。客户开户时提交的材料信息需保存到中央数据库中。为了优化数据库服务器负载，应用需要采用跨网络分布式应用，将进行远程方法调用。因此，使用 RMI 来开发此应用程序。任务具体要求如下：

- 客户开户信息包括：姓名、密码、地址、电话号码、身份证号及开户金额。
- 客户端进行远程方法调用，将用户信息发送给服务器，服务器将数据保存到数据库，并返回提示信息。

分析

1. 在 Oracle 数据库中建表。
2. 定义远程接口，提供一个添加用户信息的远程方法。
3. 实现远程接口。
4. 编写服务器类。
5. 编写客户端应用程序。
6. 编译程序代码。
7. 部署运行程序。

参考解决方案

1. 首先，使用 Scott 用户创建用户注册信息表，代码如下：

```
CREATE TABLE registration (
idcard VARCHAR2(100) primary key,
username VARCHAR2(100) NOT NULL,
pwd VARCHAR2(100) NOT NULL,
address VARCHAR2(100) NOT  NULL,
phone VARCHAR2(100) NOT NULL,
count NUMBER(10,4) NOT NULL);
```

2. 定义一个名为 Bank 的远程接口，在此接口中提供一个添加用户信息的远程方法，代码如

下所示：

```
public interface Bank extends Remote {
    String insertDetails(String userName, String pwd, String address,
            String phone, String idcard, double count) throws RemoteException;
}
```

3. 创建 BankImpl 类实现 Bank 远程接口，代码如下所示：

```
public class BankImpl extends UnicastRemoteObject implements Bank {
    protected BankImpl() throws RemoteException {
        super();
    }
    static Connection con;
    static PreparedStatement stat;
    public String insertDetails(String userName, String pwd, String address,
            String phone, String idcard, double count) throws RemoteException {
        String sReturn = "fail";
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
            stat = con.prepareStatement(
                "Insert into Registration values(?, ?, ?, ?, ?, ?)");
            stat.setString(1, idcard);
            stat.setString(2, userName);
            stat.setString(3, pwd);
            stat.setString(4, address);
            stat.setString(5, phone);
            stat.setDouble(6, count);
            int r = stat.executeUpdate();
            if (r > 0) {
                sReturn = "success";
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return sReturn;
    }
}
```

4. 编写服务器类 BankServer，代码如下所示：

```
public class BankServer {
```

```
    public BankServer() {
        try {
            Bank bank = new BankImpl();
            Context context = new InitialContext();
            context.bind("rmi://localhost:2222/BankService", bank);
            System.out.println("Server started...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String args[]) {
        new BankServer();
    }
}
```

注意　服务器类在注册表中注册远程对象时，指定了 RMI 服务端口为 2222，即 bind() 方法的第一个参数为"rmi://localhost:2222/BankService"。

5. 编写客户端类 Client，代码如下所示：

```
public class Client extends JFrame {
    JPanel panel;
    JPanel panel1;
    JLabel lblUserName;
    JLabel lblPwd;
    JLabel lblAddress;
    JLabel lblPhone;
    JLabel lblIdcard;
    JLabel lblCount;
    JTextField txtUserName;
    JPasswordField txtPwd;
    JTextField txtAddress;
    JTextField txtPhone;
    JTextField txtIdcard;
    JTextField txtCount;
    JButton btnSubmit;
    public Client() {
        super("用户开户");
        panel = new JPanel();
        panel1 = new JPanel();
        panel.setLayout(new GridLayout(6, 2));
        panel1.setLayout(new GridLayout(1, 1));
        lblUserName = new JLabel("姓名");
        lblPwd = new JLabel("密码");
        lblAddress = new JLabel("地址");
```

```
        lblPhone = new JLabel("电话");
        lblIdcard = new JLabel("身份证");
        lblCount = new JLabel("开户金额");
        txtUserName = new JTextField(30);
        txtPwd = new JPasswordField(30);
        txtAddress = new JTextField(30);
        txtPhone = new JTextField(30);
        txtIdcard = new JTextField(30);
        txtCount = new JTextField(30);
        btnSubmit = new JButton("提交");
        panel.add(lblUserName);
        panel.add(txtUserName);
        panel.add(lblPwd);
        panel.add(txtPwd);
        panel.add(lblAddress);
        panel.add(txtAddress);
        panel.add(lblPhone);
        panel.add(txtPhone);
        panel.add(lblIdcard);
        panel.add(txtIdcard);
        panel.add(lblCount);
        panel.add(txtCount);
        panel1.add(btnSubmit);
        ButtonListener blisten = new ButtonListener();
        btnSubmit.addActionListener(blisten);
        this.add(new JPanel(), BorderLayout.WEST);
        this.add(new JPanel(), BorderLayout.EAST);
        this.add(panel, BorderLayout.CENTER);
        this.add(panel1, BorderLayout.SOUTH);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400, 200);
        this.setVisible(true);
    }
    class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent evt) {
            String userName = txtUserName.getText();
            String pwd = new String(txtPwd.getPassword());
            String address = txtAddress.getText();
            String phone = txtPhone.getText();
            String idcard = txtIdcard.getText();
            String count = txtCount.getText();
            // 初始验证
            if (userName.equals("")) {
                JOptionPane.showMessageDialog(btnSubmit,"用户名不能为空","提示",
```

```
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (pwd.equals("")) {
            JOptionPane.showMessageDialog(btnSubmit,"密码不能为空","提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (address.equals("")) {
            JOptionPane.showMessageDialog(btnSubmit, "地址不能为空", "提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (phone.equals("")) {
            JOptionPane.showMessageDialog(btnSubmit,"电话不能为空","提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (idcard.equals("")) {
            JOptionPane.showMessageDialog(btnSubmit,"身份证不能为空","提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (count.equals("")) {
            JOptionPane.showMessageDialog(btnSubmit,"金额不能为空","提示",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        try {
            // 声明全局注册表
            Context context = new InitialContext();
            // 查找 RMI 服务中的服务对象
            Bank bank = (Bank)
                 context.lookup("rmi://localhost:2222/BankService");
            String str = bank.insertDetails(userName, pwd, address, phone,
                    idcard, Double.parseDouble(count));
            System.out.println(str);
            if (str.equals("success")) {
            JOptionPane.showMessageDialog(btnSubmit,"开户成功！","提示",
                    JOptionPane.PLAIN_MESSAGE);
            } else {
            JOptionPane.showMessageDialog(btnSubmit,"开户失败！","提示",
                    JOptionPane.PLAIN_MESSAGE);
            }
```

```
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
    public static void main(String args[]) {
        new Client();
    }
}
```

注意 lookup()方法中需指明 RMI 服务端口为 2222。

6. 在 Windows 系统下，通过命令行模式进入 RMI 源文件所在文件夹，使用下面命令：

```
javac *.java
```

编译程序代码，如图 7.1 所示。

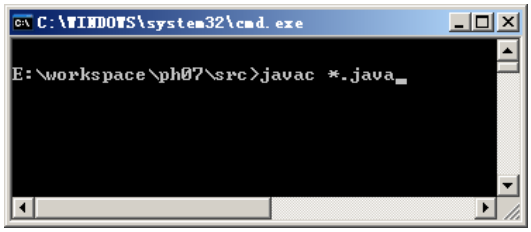


图 7.1 编译代码

7. 启动注册表，命令如下：

```
start rmiregistry 2222
```

注意 为启动不同端口处的 RMI 注册表，需在命令行中指明端口号。本例启动 2222 端口处的注册表（与服务器和客户端程序代码中的端口号一致）。

8. 启动服务器，命令如下：

```
start java BankServer
```

服务器启动如图 7.2 所示。

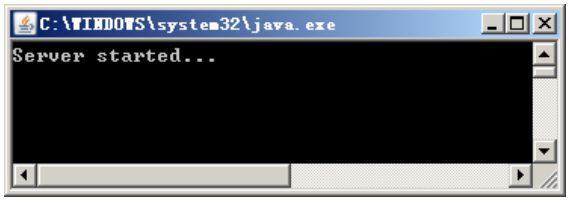


图 7.2 启动服务器

9. 运行客户端应用程序，命令如下：

```
start java Client
```

显示客户端图形界面，如图 7.3 所示。

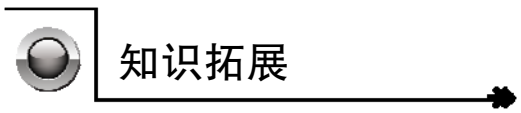


图 7.3　客户端图形界面

输入用户开户信息，当开户成功则提示成功信息，如图 7.4 所示。



图 7.4　提示信息



1. CORBA 技术

CORBA（Common Object Request Broker Architecture）是公共对象请求代理体系结构，它提供了分布、异构对象之间的通信标准架构。它是由对象管理团体（OMG）定义的，该团体大约由七百个组织和公司组成，其中分布式技术的主要厂商和开发者都是此团体的成员。CORBA 和 RMI 都是为分布式应用的中间层上的远程对象通信而定义的。但 RMI 是 JDK 提供的一种设施，捆绑在 Java 平台上；而 CORBA 提供了多个平台的可移植的解决方案，不管是用什么语言或操作系统。RMI 与 CORBA 的另一个区别是：RMI 按值传参数，而 CORBA 按引用传递参数。

CORBA 架构由以下构件组成：

- 对象请求代理（ORB）
- 公共对象服务

- 接口定义语言（IDL）
- ORB 之间的协议
- 基本对象适配器（BOA）

其中，ORB（Object Request Broker）是 CORBA 的基石，是客户端与服务器之间的中间件，提供了客户端远程与服务器通信的异构环境，不受所用的硬件、操作系统或软件开发语言的影响。CORBA 的组成见图 7.5，该图表示了客户、服务的执行对象与 ORB 接口间的关系。

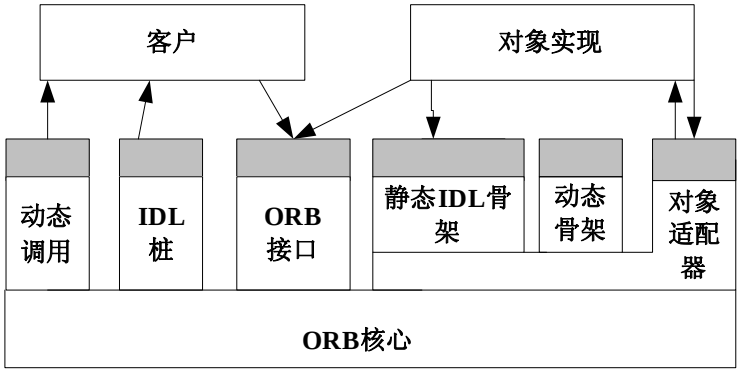


图 7.5　CORBA 组成

IDL 不是任何特定的语言，它可用来声明方法与数据类型，是与 ORB 通信的接口。IDL 定义的接口经 IDL 编译器编译后能产生客户桩（stub）及执行对象的骨架（skeleton）。其中，客户桩（stub）是在编译时确定的静态接口。它位于客户对象的本地，接受客户的请求，对客户来说相当于远程的执行对象。动态调用接口（DII）用于编译时不能确定的请求，与客户桩（stub）的作用相同。骨架（Skeleton）分动态骨架与静态骨架，它代表了远程客户的作用，可在本地调用执行对象服务，并与 ORB 通信，如图 7.6 所示。

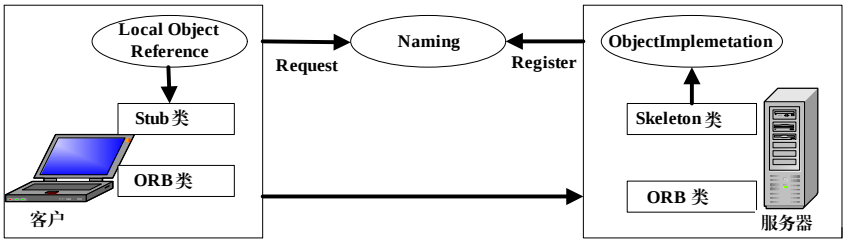


图 7.6　通信过程

对象适配器提供执行对象可以访问 ORB 服务的基本接口，其作用是产生及解释对象引用、安全交互、登记和执行等。

CORBA 作为标准化接口的规范及体系结构正在发展中，最初仅考虑了基本机制，但随着对异构分布应用互操作性的深入认识，不断有人提出 CORBA 应提供更多的服务要求 RFP

（Request for Proposal），如命名服务、交易服务和生命周期等。CORBA 2.1 中都已纳入以上服务要求。

开发 CORBA 应用程序主要有以下几个步骤：

01 定义 IDL 接口。
02 将 IDL 转换成 Java。
03 应用接口。
04 开发服务器端。
05 开发客户端。
06 运行名称服务器、服务器和客户程序。

在此就不具体举例了。

2. SOAP

SOAP（Simple Object Access Protocol）是简单对象访问协议，它是一种轻量的、简单的、基于 XML 的协议，被设计成在 Web 上交换结构化的和固化的信息。SOAP 可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（HTTP）、简单邮件传输协议（SMTP）、多用途网际邮件扩充协议（MIME）。它还支持从消息系统到远程过程调用（RPC）等大量的应用程序。

SOAP 包括以下 3 个部分：

- SOAP 封装：定义了一个框架，该框架描述了消息中的内容是什么，谁应当处理及是可选的还是必需的。
- SOAP 编码规则：定义了一种序列化的机制，用于交换应用程序所定义的数据类型的实例。
- SOAP RPC 表示：定义了用于表示远程过程调用和应答的协定。

SOAP 消息格式如下：

```
SOAP 标头
<SOAP-ENV: Envelope Attributes>
<SOAP-ENV:Body Attributes>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP 消息的规范如下：

- SOAP 消息必须用 XML 来编码。
- SOAP 消息必须使用 SOAP Envelope 命名空间。
- SOAP 消息不能包含 DTD 引用。

- SOAP 消息不能包含 XML 处理指令。



拓展练习

练习 7.E.1

编写远程 RMI 应用程序：在客户端，用户可以选择不同的商品名称；服务器提供查询商品详细信息的功能，编写服务器类和客户类的代码。

练习 7.E.2

基于 7.G.1，使用远程 RMI 实现用户登录查询账户余额的功能。

附录 A　正则表达式元字符

字符	描述			
\	将下一个字符标记为一个特殊字符、一个原义字符、一个后向引用或一个八进制转义符。例如，'n' 匹配字符 "n"，'\n' 匹配一个换行符，序列 '\\' 匹配 "\"，"\(" 则匹配 "("			
^	匹配输入字符串的开始位置			
$	匹配输入字符串的结束位置			
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 及 "zoo"。 * 等价于{0,}			
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}			
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do"。? 等价于 {0,1}			
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o			
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'，'o{0,}' 则等价于 'o*'			
{n,m}	m 和 n 均为非负整数，其中 n≤m。最少匹配 n 次且最多匹配 m 次。如， "o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格			
?	当该字符紧跟在其他任何一个限制符（*, +, ?, {n}, {n,}, {n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少地匹配所搜索的字符串，而默认的贪婪模式则尽可能多地匹配所搜索的字符串。例如，对于字符串"oooo"，'o+?' 将匹配单个"o"，而'o+'将匹配所有'o'			
.	匹配除"\n"之外的任何单个字符。要匹配包括'\n'在内的任何字符，请使用如'[.\n]'的模式			
(pattern)	匹配 pattern 并获取这一匹配			
(?:pattern)	匹配 pattern 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符（	）来组合一个模式的各个部分是很有用。例如， 'industr(?:y	ies) 就是一个比 'industry	industries' 更简略的表达式
(?=pattern)	正向预查，在任何匹配 pattern 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如， 'Windows (?=95\|98\|NT\|2000)' 能匹配 "Windows 2000" 中的 "Windows" ，但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始			

（续表）

字符	描述
(?!pattern)	负向预查，在任何不匹配的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如'Windows (?!95\|98\|NT\|2000)' 能匹配 "Windows 3.1" 中的 "Windows"，但不能匹配 "Windows 2000" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
x\|y	匹配 x 或 y。例如，'z\|food' 能匹配 "z" 或 "food"。'(z\|f)ood' 则匹配 "zood" 或 "food"
[xyz]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'
[^xyz]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'
[a-z]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'
\cx	匹配由 x 指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。 x 的值必须为 A~Z 或 a~z 之一，否则，将 c 视为一个原义的 'c' 字符
\d	匹配一个数字字符，等价于 [0-9]
\D	匹配一个非数字字符，等价于 [^0-9]
\f	匹配一个换页符，等价于 \x0c 和 \cL
\n	匹配一个换行符，等价于 \x0a 和 \cJ
\r	匹配一个回车符，等价于 \x0d 和 \cM
\s	匹配任何空白字符，包括空格、制表符、换页符等，等价于[\f\n\r\t\v]
\S	匹配任何非空白字符，等价于[^ \f\n\r\t\v]
\t	匹配一个制表符，等价于\x09 和\cI
\v	匹配一个垂直制表符，等价于\x0b 和\cK
\w	匹配包括下画线的任何单词字符，等价于'[A-Za-z0-9_]'
\W	匹配任何非单词字符，等价于 '[^A-Za-z0-9_]'
\xn	匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，'\x41' 匹配"A"。'\x041'则等价于'\x04' & "1"。正则表达式中可以使用 ASCII 编码
\num	匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，'(.)\1'匹配两个连续的相同字符

（续表）

字符	描述
\n	标识一个八进制转义值或一个后向引用。如果\n 之前至少 n 个获取的子表达式，则 n 为后向引用。如果 n 为八进制数字(0-7)，则 n 为一个八进制转义值
\nm	标识一个八进制转义值或一个后向引用。如果\nm 之前至少有 nm 个获取的子表达式，则 nm 为后向引用。如果\nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的后向引用。如果前面的条件都不满足，若 n 和 m 均为八进制数字（0~7），则\nm 将匹配八进制转义值 nm
\nml	如果 n 为八进制数字（0~3），且 m 和 l 均为八进制数字（0~7），则匹配八进制转义值 nml
\un	匹配 n，其中 n 是一个用 4 个十六进制数字表示的 Unicode 字符。例如，\u00A9 匹配版权符号(?)